Extensive Study of Learners for Defect Prediction

Amritanshu Agrawal
Department of Computer Science
North Carolina State University
Raleigh, NC, USA
Email: aagrawa8@ncsu.edu

Raghavendra Prasad Potluri
Department of Computer Science
North Carolina State University
Raleigh, NC, USA
Email: rpotlur@ncsu.edu

Abstract—**The accurate prediction of where faults are likely to occur in code can help direct test effort, reduce costs and improve the quality of software. We are reviewing how the context of models, the independent variables used and the modelling techniques applied, influence the performance of fault prediction models. We surveyed 8 papers published from 2009 to December 2016 considering a baseline paper from Hall et al. [8] After this study, we found out that the learners, like Random Forest, Decision trees, Simple Logistic, Naive Bayes and nearest neighbour outperformed all other learners. Combinations of independent variables have been used by models that perform well. Feature selection has also been applied to these combinations when models are performing particularly well. The methodology used to build models seems to be influential to predictive performance. Although there are a set of fault prediction studies in which confidence can possibly be more.**

Keywords—Defects, prediction, code metrics, classification.

I. Introduction

Software defect prediction has been an important research topic in the software engineering field for more than 30 years. Current defect prediction work focuses on (i) estimating the number of defects remaining in software systems, (ii) discovering defect associations, and (iii) classifying the defect-proneness of software components, typically into two classes defect-prone and not defect-prone. This survey tried to find details on all the above mentioned work.

The first type of work employs statistical approaches, capture-recapture (CR) models, and detection profile methods (DPM) [22]. The second type of work borrows association rule mining algorithms from the data mining community to reveal software defect associations [23]. A variety of approaches have been proposed to tackle the third type of problem, relying on diverse information, such as code metrics [19] (lines of code, complexity), process metrics [9] (number of changes, recent activity) or previous defects [11].

Some other research [2] indicate that it is possible to predict which components are likely locations of defect occurrence using a components development history, and dependency structure. Two key properties of software components in large systems are dependency relationships (which components depend on or are dependent on by others), and development history (who made changes to the components and how many times). Thus we can link software components to other components a) in terms of their dependencies, and also b) in terms of the developers that they have in common. Prediction models based on the topological properties of components within them have proven to be quite accurate [27].

There have been vast amount of studies done to find the best defect prediction performing model. But literature suggests, that no single prediction technique dominates and making sense of the many prediction results is hampered by the use of different data sets, data pre-processing, validation schemes and performance statistics. We highly agree to this given so many variations available in the data and there are so many classification techniques available like Statistical, Clustering, Rule-Based, Neural Networks, Nearest Neighbour, Support Vector Machines, Decision trees, ensemble methods, to name a few.

Result by Tantithamthavorn et al. [25] also suggested that every dataset comes with different attributes. And also classification techniques often have configurable parameters that control characteristics of these classifiers that they produce. Now time has come to even think about hyperparameter optimization of these techniques and come up with an automated process [6], [1] to tune these parameters for every dataset.

The remainder of this review is organized as follows. Section II talks about how the survey is done. Section III talks about papers till 2012. Section IV talks about papers after 2012. Section V finally gives a conclusion and discussion on our views.

II. Survey Overview

Note that the organization of the following sections occurs chronologically, beginning with papers published prior to 2012, follow by a summary of the paper by Hall et al. [8] on fault prediction, and concluding with papers published in 2013 or later.

III. Survey Approaching 2012

The following papers were composed before Hall et al. [8] published on the Systematic literature review on fault prediction performance in Software Engineering in 2012. For each paper, we provide a bullet point format, defining keywords described by each author, summarizing selected ideas presented in each paper, and finally, the conclusion and describing what can be made to improve the results.

A. ***Putting it All Together: Using Socio-Technical Networks to Predict Failures [2]***

1) ***Keywords:***

- SNA: Social network analysis (SNA) is the process of investigating social structures through the use of network and graph theories.

- Contribution network: Contribution network captures the contributions of developers to software components within the system.
- Dependency network: Dependency network models the dependency relationships between the software components within the system.
- Socio-technical Network: A socio-technical network is created by combining dependency and contribution relationships into on graph. Both of the above networks deal with information & control flow. The joint network then, captures the interaction between the two.

*2) **Key Ideas**:*

- Motivation: Task assignment (i.e. who worked on which components and how much) and dependency structure (which components have dependencies on others) together interact to influence the quality of the resulting software. It can be very difficult and expensive to test all of the components of a large and complex system. However, the complexity inherent in large software systems can be leveraged to aid in locating those components which are particularly defect prone. Components which play key roles and are central in these networks tend to be more failure prone than components in the surrounding areas.
- Related Work: Authors found that models built on social network metrics were better indicators of future failures than models based on standard source code metrics. This approach leveraged SNA metrics to capture both local and global effects of network connectivity on defect-proneness. Software artifacts such as email interactions, and developers contribution history have also influenced SNA. That global connectivity measures such as *betweenness* were better indicators of development activity than local measures such as *degree centrality*. But by studying contribution history, degree centrality, closeness centrality, and Bonacich power have very good predictive power.
- Patterns: Software components that play key roles in the joint socio-technical network are more prone to defects than those that don't. Logistic regression to examine the relationship between social network analysis metrics and post-release failures. They used PCA to reduce the dimensionality of feature space
- Assessment: They collected the data and assessed the results in both a traditional industrial setting (Windows Vista) and an open source software (OSS) setting (Eclipse). They used global measures examine the position of the component within the context of the entire network and include betweenness, Bonacich Power, and eigenvector centrality. Local measures only take into account the neighborhood of nodes within one or two hops of software component. These include measures such as degree, size of the network, and edge density. Broad description on these measures are available in this paper. And to evaluate their prediction results, they reported precision, recall and fscore.

*3) **Conclusion and Improvement**:* They found out that neither the dependency network model nor the contribution network model were superior to either the combined or socio-technical models for any of the evaluation metrics. The recall for the combined and socio-technical models exceeds the previous work by 3% and 5% respectively which is substantial given the thousands of samples are made. There are scopes of improvement in following ways:

- They only studied 2 projects, one from an industrial setting and another from open source community, but we can argue that these results might be sample biased. It can still get affected from external validity. More datasets results should be reported.
- It would be great to see the results of defect predictions using combined and socio technical network together.

B. An extensive comparison of bug prediction approaches [5]

*1) **Keywords**:*

- Change Log Approaches: It uses information extracted from the versioning system, assuming that recently or frequently changed files are the most probable source of future bugs.
- Single-version approaches: It assumes that the current design and behavior of the program influences the presence of future defects. These approaches do not require the history of the system, but analyze its current state in more detail, using a variety of metrics.
- Principal Component Analysis: Principal component analysis (PCA) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components. The number of principal components is less than or equal to the number of original variables.
- Exponentially Decayed HCM, every file modified in the considered period i gets the entropy of the system in the considered time interval.

*2) **Key Ideas**:*

- Motivation: The driving scenario is resource allocation: Time and manpower being finite resources, it makes sense to assign personnel and/or resources to areas of a software system with a higher probable quantity of bugs. They want to make it more automated so that it takes less man power.
- Related Work: To tackle there were many approaches suggested code metrics (lines of code, complexity), process metrics (number of changes, recent activity) or previous defects. Relative code churn was a better predictor than absolute churn. The bug-introducing changes are identified from the SCM logs. Comparative study among process metrics, system metrics, defect information related and bi-weekly models of each system version if new metrics need to be computed. Entropy was compared to amount of changes and the amount of previous bugs. Chidamber and Kemerer

	Adjusted R^2 - Explanative power						Spearman correlation - Predictive power					
Predictor	Eclipse	Mylyn	Equinox	PDE	Lucene	Score	Eclipse	Mylyn	Equinox	PDE	Lucene	Score
Change metrics (Section IV-A)												
MOSER	0.454	**0.206**	0.596	0.517	**0.57**	9	0.323	**0.284**	**0.534**	0.165	0.238	6
NFIX-ONLY	0.143	0.043	0.421	0.138	0.398	-3	0.288	0.148	0.429	0.113	0.284	-1
NR	0.38	0.128	0.52	0.365	0.487	2	0.364	0.099	**0.548**	0.245	0.296	5
NFIX+NR	0.383	0.129	0.521	0.365	0.459	2	0.381	0.091	**0.567**	0.255	0.277	4
Previous defects (Section IV-B)												
BF (short for BUGFIXES)	0.487	0.161	0.503	0.539	**0.559**	5	**0.41**	0.159	0.492	**0.279**	**0.377**	10
BUG-CAT	0.455	0.131	0.469	0.539	**0.559**	5	**0.434**	0.131	0.513	**0.284**	**0.353**	9
Source code metrics (Section IV-C)												
CK+OO	0.419	0.195	**0.673**	**0.634**	0.379	8	0.39	**0.299**	0.453	**0.284**	0.214	8
CK	0.382	0.115	0.557	0.058	0.368	0	0.377	0.226	0.484	0.256	0.216	4
OO	0.406	0.17	**0.619**	**0.618**	0.209	6	0.395	**0.297**	0.49	0.263	0.214	6
LOC	0.348	0.039	0.408	0.04	0.077	-3	0.38	0.222	0.475	0.25	0.172	2
Entropy of changes (Section IV-D)												
HCM	0.366	0.024	0.495	0.13	0.308	-2	**0.416**	-0.001	0.526	0.244	0.308	5
WHCM	0.373	0.038	0.34	0.165	0.49	-1	**0.401**	0.076	**0.533**	0.273	0.288	7
EDHCM	0.209	0.026	0.345	0.253	0.22	-4	0.371	0.07	0.495	0.258	0.306	3
LDHCM	0.161	0.011	0.463	0.267	0.216	-4	0.377	0.064	**0.581**	**0.28**	0.275	6
LGDHCM	0.054	0	0.508	0.209	0.141	-3	0.364	0.03	**0.562**	0.263	0.33	5
Churn of source code metrics (Section IV-E)												
CHU	0.445	0.169	**0.645**	**0.628**	0.456	8	0.371	0.226	0.51	0.251	0.292	5
WCHU	**0.512**	0.191	**0.645**	**0.608**	0.478	11	**0.419**	**0.279**	**0.56**	**0.278**	0.285	13
LDCHU	**0.557**	**0.214**	0.581	**0.616**	0.458	11	0.395	**0.275**	**0.563**	**0.307**	0.293	11
EDCHU	**0.509**	**0.227**	0.525	**0.598**	0.467	11	0.362	0.259	0.464	**0.294**	0.28	6
LGDCHU	0.473	0.095	**0.642**	0.486	0.493	5	**0.442**	0.188	**0.566**	0.189	0.29	7
Entropy of source code metrics (Section IV-F)												
HH	0.484	0.199	**0.667**	0.514	0.433	7	**0.405**	**0.277**	0.484	0.266	0.318	9
HWH	0.473	0.146	**0.621**	**0.641**	0.484	8	**0.425**	0.212	0.48	0.266	0.263	5
LDHH	**0.531**	**0.209**	0.596	0.522	0.343	8	**0.408**	**0.272**	**0.53**	**0.296**	0.333	13
EDHH	0.485	**0.226**	0.469	0.515	0.359	5	0.366	**0.273**	**0.586**	**0.304**	0.337	11
LGDHH	0.479	0.13	**0.66**	0.447	0.419	4	**0.421**	0.185	0.492	0.236	**0.347**	8
Combined approaches												
BF+CK+OO	0.492	**0.213**	**0.707**	**0.649**	**0.586**	13	**0.439**	**0.277**	**0.547**	**0.282**	**0.362**	15
BF+WCHU	**0.536**	0.193	**0.645**	**0.627**	**0.594**	13	**0.448**	0.265	**0.533**	**0.282**	0.31	11
BF+LDHH	**0.561**	**0.217**	**0.615**	**0.601**	**0.592**	15	**0.422**	0.221	**0.533**	**0.305**	**0.352**	12
BF+CK+OO+WCHU	**0.559**	**0.25**	**0.734**	**0.661**	**0.61**	15	**0.425**	**0.306**	0.524	**0.31**	0.298	11
BF+CK+OO+LDHH	**0.587**	**0.262**	**0.73**	**0.68**	**0.618**	15	**0.44**	**0.291**	**0.571**	**0.312**	**0.377**	15
BF+CK+OO+WCHU+LDHH	**0.62**	**0.277**	**0.754**	**0.691**	**0.65**	15	**0.408**	**0.326**	**0.592**	**0.289**	**0.341**	15

Table IV
EXPLANATIVE AND PREDICTIVE POWER FOR ALL THE APPROACHES.

(CK) metrics suite, McCabes cyclomatic complexity, Briands coupling metrics, code metrics, dependencies between binaries, cohesion measurement based on LSI.

- Patterns: Predictions at the package-level are less helpful since packages are significantly larger. Package-level information can be derived from class-level information, while the opposite is not true. They used PCA, built regression model. Approaches based on churn and entropy of source code metrics have good and stable explanative and predictive power, better than all the other applied approaches. Using the source code metrics, CK+OO to predict bugs has several advantages: They are lightweight to compute, have good explanative and predictive power and do not require historical information.
- Results: Please refer Table IV.
- Datasets used : Six open-source systems: FreeBSD, NetBSD, OpenBSD, KDE, KOffice, and PostgreSQL. Apache, PostgreSQL, Subversion, Mozilla, JEdit, Columba, and Eclipse. Datasets publicly available here[1].

*3) **Conclusion and improvement**:* Using the CK and the OO metric sets together is preferable than using them in isolation, as the performances are more stable across case studies. Bug prediction approaches based on a single metric are not stable over the case studies. The best weighting for past metrics is the linear one. Using string matching on versioning system comments, without validating it on the bug database, decreases the accuracy of bug prediction. Combining bugs and OO metrics improves predictive power. There are scopes of improvement in following ways:

- Set of bugs which are linked to commit comments is not a fair representation of the full population of bugs. Need to have better sampling technique.
- Considerable fraction of problem reports marked as bugs in Bugzilla (according to their severity) are indeed ”non bugs”. Need an automated technique to find bugs or not bugs.

C. A general software defect-proneness prediction framework [22]

*1) **Keywords**:*

- InfoGain: The expected information gain is the change in information entropy H from a prior state to a state that takes some information.
- Forward selection: It starts from an empty set and evaluates each attribute individually to find the best single attribute. It then tries each of the remaining attributes in conjunction with the best to find the best pair of attributes. In the next iteration each of the

[1]http://bug.inf.usi.ch

remaining attributes are tried in conjunction with the best pair to find the best group of three attributes.

- Backward elimination: It starts with the whole set of attributes, and eliminates one attribute in each iteration until no single attribute elimination improves the evaluation of the subset.
- Wilcoxon signed-rank test: It is a non-parametric statistical hypothesis test used when comparing two related samples, matched samples, or repeated measurements on a single sample to assess whether their population mean ranks differ. It can be used as an alternative to the paired Student's t-test, t-test for matched pairs, or the t-test for dependent samples when the population cannot be assumed to be normally distributed.

*2) **Key Ideas**:*

- Motivation: Different learning schemes are needed for different data sets (i.e. no scheme dominates), that small details in conducting how evaluations are conducted can completely reverse findings and lastly that their proposed framework is more effective, and less prone to bias than previous approaches.
- Related Work: Capture-recapture (CR) models and detection profile methods (DPM) to estimate the number of defects remaining in software systems with inspection data and process quality data. Association rule mining algorithms reveal software defect associations. Other work classifies software components as defect-prone and non-defect-prone by means of metric-based classification. Their work is mostly inspired by the work done by Menzies et al. [16]. They compared Rule Induction and Naive Bayes to predict software components containing defects.
- Patterns: The framework should consist of two components: (i) scheme evaluation and (ii) defect prediction. Scheme evaluation will evaluate the machine learning algorithm (which performs better with respect to historical data) before using it for defect prediction. At the first stage, attribute selection is needed that can be categorized as either filters or wrappers and only used on the training set. There is slight difference between Menzies work and this work. Just the order in which cross validation, attribute selection and wrapping filters are applied. Defect prediction with different learning schemes. In short there are 12 comparisons based learning which includes 2 data preprocessors, two attribute selectors, and three learning schemes.
- Datasets used : The public NASA MDP repository, which was also used by Menzies et al. Whats more, the AR data from the PROMISE repository[2] was also used. Thus there are 17 data sets in total, 13 from NASA and the remaining 4 from the PROMISE repository.

*3) **Conclusion and improvement**:* They observed that there is a bigger difference between the evaluation performance and the actual prediction performance in Menzies et al. study than with their framework. This means that the results reported by Menzies et al. are over optimistic. They contend that their framework is less biased and more capable of yielding results closer to the true answer. Moreover, their framework is more stable. There are scopes of improvement in following ways:

- They should report the results by better statistical tests like scott-knot test.
- Data preprocessor/attribute selector can play different roles and they only reported the results with only couple of options.
- They surely reported results with 3 different learners, but as they stated these can be quite learner biased. We will need to run many different learners for different datasets.

D. A systematic literature review on fault prediction performance in software engineering [8]

*1) **Keywords**:*

- Systematic Literature Review: A systematic review is a type of literature review that collects and critically analyzes multiple research studies or papers.
- Mean Standard Error: The MSE is the standard deviation of the sample-mean's estimate of a population mean.
- Area Under Curve: The area under a curve between two points can be found by doing a definite integral between the two points. To find the area under the curve y = f(x) between x = a and x = b, integrate y = f(x) between the limits of a and b.
- Receiver operating characteristic Curve: A graphical plot of the sensitivity (or pd) vs. 1 specificity (or pf) for a binary classification system where its discrimination threshold is varied

*2) **Key Ideas**:*

- Motivation : Context is important in fault prediction modeling as it can affect the performance of models in a particular context and the transferability of models between contexts. There are a range of independent variables that have been used in fault prediction models. Fault prediction models are based on a wide variety of both machine learning and regression model techniques. Currently the impact context have on transferability of models, the impact individual independent variables have on model performance and the impact modelling technique has on model performance is not clear. This makes it difficult for model builders to make effective technique selections. Hall et al. aims to present a synthesis of current knowledge on the impact of context, independent variables and model techniques on model performance.
- Sampling Procedures: A set of 208 studies performed addressing fault prediction in software engineering from January 2000 to December 2010. They defined various criteria to come up with those papers that can

[2]4. http://promise.site.uottowa.ca/SERepository

be of utmost importance. They defined the paper's inclusion and exclusion criteria, Paper selection and validation process, Prediction criteria, context criteria, model building criteria, data criteria.

- Patterns: The context of models has not been studied extensively in the set of studies they analysed. This is a significant gap in current knowledge as it means we currently do not know what context factors influence how well a model will transfer to other systems. Models using only static code metrics (typically complexity-based) perform relatively poorly. Model performance does not seem to improve by combining these metrics with OO metrics. Models seem to perform better using only OO metrics. The use of process data is not particularly related to good predictive performance. Analysis suggests that studies using Support Vector Machine (SVM) techniques perform less well. These may be underperforming as they require parameter optimization. Models based on C4.5 seem to underperform if they have imbalanced data. Naive Bayes and Logistic regression, in particular, seem to be the techniques used in models that are performing relatively well.

- Anti-Patterns: Prediction shouldn't be performed if data quality is poor. Good data quality is important for prediction. Data should not be imbalanced. There is no one best way to measure the performance of a model. Performance comparison across studies is only possible if studies report a set of uniform measures. Companies developing non-critical systems may want to prioritise their fault finding effort only on the most severe faults.

*3) **Conclusion and improvement:*** Their results suggest that models which perform well tend to be built in a context where the systems are large. We found no evidence that the maturity of systems or the language used is related to predictive performance. Overall they conclude that many good fault prediction studies have been reported in software engineering. There are scopes of improvement in following ways:

- We need more studies which are based on a reliable methodology and which consistently report the context in which models are built and the methodology used to build them.

- A larger set of such studies will enable reliable cross-study metaanalysis of model performance. It will also give practitioners the confidence to appropriately select and apply models to their systems.

- Cross project defect/fault prediction can be done too, where privacy of data is of concern.

- Hyperparameter optimization of learners should be done.

- For imbalanced dataset, SMOTE [4] could be used.

IV. Survey After 2012

The following papers were comprises after Hall et al. [8] published on the Systematic literature review on fault prediction performance in Software Engineering in 2012. For each paper, we provide a bullet point format, defining keywords described by each author, summarizing selected ideas presented in each paper, and finally, the conclusion and describing what can be made to improve the results.

A. *Software fault prediction metrics: A systematic literature review [20]*

*1) **Keywords:***

- Software metric: A software metric is a standard of measure of a degree to which a software system or process possesses some property.

- MOOD metrics suite: The MOOD metrics suite includes 6 metrics: Method Hiding Factor, Attribute Hiding Factor, Method Inheritance Factor, Attribute Inheritance Factor, Polymorphism Factor, and Coupling Factor.

- CK metrics suite: The CK metrics suite includes 6 metrics: Weighted Method Class, Depth of Inheritance tree, Number of children, Coupling between Object Classes, Response for a class, and Lack of cohesion in Methods.

*2) **Key Ideas:***

- Motivation: In software fault prediction many software metrics have been proposed. Many of them have been validated only in a small number of studies. Some of them have been proposed but never used. Contradictory results across studies have been reported. Even withing a single study, different results have been obtained when different environments or methods have been used. The aim of the authors were to depict current state-of-the-art metrics in software fault prediction.

- Related Work: A systematic review of software fault prediction studies was performed by Catal et al [3]. In 2012, a review similar in design to Catal et al, but more comprehensive in terms of the number of included studies and analyses, was published by Hall et al [8]. In the review, papers on software fault prediction were included focusing again on empirical studies. This paper is different from the above reviews in both the aim and scope of selected studies. The objective of this review are to asses primary studies that empirically validate software metrics in software fault prediction and to assess metrics used in these studies according to several properties.

- Patterns: Validations should be performed in the most realistic environment possible in order to acquire results relevant for the industry. In realistic environments, faults are fairly random and data sets are highly unbalanced (few faults, many correct modules). The number of faults and their distribution between two releases can be significantly different. Validation techniques, like a 10-fold cross-validation, 2/3 for training and 1/3 for testing, do not take into account all the factors that real environment validation does. Only validation where models are trained on release i and evaluated on i+1, can determine the impact of all these, and other unpredictable factors, of the environment.

- Sampling Procedures: A set of 106 primary studies evaluating software metrics performed addressing fault prediction in software engineering. They defined various criteria to come up with those papers that can be of utmost importance. They defined the paper's inclusion and exclusion criteria, Paper selection and validation process, Prediction criteria, context criteria, model building criteria, data criteria.

3) ***Conclusion and improvement:*** Object-oriented metrics (49%) were used nearly twice as often compared to traditional source code metrics (27%) or process metrics (24%). Chidamber and Kemerer's (CK) objected-oriented metrics were most frequently used. According to the selected studies there are significant differences between the metrics used in fault prediction performance. Objected-oriented and process metrics have been reported to be more successful in finding faults faults compared to any traditional size and complexity metrics. There are scopes of improvement in following ways:

- More studies should be performed on large industrial software systems to find metrics more relevant for the industry and to answer the question as to which metrics should be used in a given context.

B. ***Researcher Bias: The Use of Machine Learning in Software Defect Prediction [21]***

1) ***Keywords:***

- Software defect prediction: Software Defect Prediction (SDP) is one of the most assisting activities of the Testing Phase of SDLC. It identifies the modules that are defect prone and require extensive testing. This way, the testing resources can be used efficiently without violating the constraints.
- Meta-Analysis: In statistics, meta-analysis comprises statistical methods for contrasting and combining results from different studies, in the hope of identifying patterns among study results, sources of disagreement among those results, or other interesting relationships that may come to light in the context of multiple studies.
- Researcher Bias: Research bias, also called experimenter bias, is a process where the scientists performing the research influence the results, in order to portray a certain outcome.
- Matthews correlation coefficient: MCC is used in machine learning as a measure of the quality of binary (two class) classifications.

2) ***Key Ideas:***

- Motivation : The ability to predict defect-prone software components would be valuable. Consequently, there have been many empirical studies to evaluate the performance of different techniques endeavouring to accomplish this effectively. However, no one technique dominates and so designing a reliable defect prediction model remains problematic. They seek to make sense of the many conflicting experimental results and understand which factors have the largest effect on predictive performance.
- Related Work: There has been very substantial research effort put into software defect prediction. However, a consensus on what are the best prediction techniques, even for a specific context, remains elusive. This paper finds considerable diversity as to what form of classifier technique should be used and what inputs or metrics work best. In addition, in order to facilitate generalisation, researchers are using an increasing number of different defect data sets for empirical validation. Unfortunately the situation is somewhat complicated by the use of a wide range of validation schemes. The time is ripe to explore the underlying reasons for this lack of convergence in results hence this paper conducted a meta-analysis. This will provide pointers to the most effective way forward for future software defect prediction research.
- Sampling Procedures: They conducted a meta-analysis of all relevant, high quality primary studies of defect prediction to determine what factors influence predictive performance. This is based on 42 primary studies that satisfy our inclusion criteria that collectively report 600 sets of empirical prediction results.
- Patterns: There is no uniformity amongst researchers as to how classification performance should be reported. This lack of uniformity has necessitated them to reverse engineer a common performance statistic (the Matthews correlation coefficient (MCC). By reverse engineering a common response variable, they built a random effects ANOVA model to examine the relative contribution of four model building factors (classifier, data set, input metrics and researcher group) to model prediction performance. The number of observations of the correlation coefficient is close to zero or even negative. This reveals that many classifiers are performing extremely poorly, since zero indicates no relationship at all and could therefore be achieved through guessing. A negative value means the prediction would be improved by adding an inverter. Moreover, the modal value lies in the range of 0.3-0.4 which hardly engenders a great deal of confidence in terms of their practical use.

3) ***Conclusion and improvement:*** The variability due to the choice of classifier is extremely small. The variability due to the groups of researchers (Researcher Group) could be seen as twenty five times higher. There are differing levels of expertise between researcher groups and second, there is the widespread problem of researcher bias. There are scopes of improvement in following ways:

- Improving communication and documentation will help protect against the other things which researcher groups are doing, i.e. the unwritten setup is just as important as that which is documented.
- Joint comparative empirical studies of defect prediction between research centres such that researcher

groups do not have to be expert in all classifier learning techniques.

C. *Revisiting the impact of classification techniques on the performance of defect prediction models [7]*

1) Keywords:

- MARS: Multivariate Adaptive Regression Splines is a non-parametric regression technique and can be seen as an extension of linear models that automatically models nonlinearities and interactions between variables.
- LMT: logistic model tree (LMT) is a classification model with an associated supervised training algorithm that combines logistic regression (LR) and decision tree learning.
- Expectation Maximization: An EM algorithm is an iterative method for finding maximum likelihood or maximum a posteriori (MAP) estimates of parameters in statistical models, where the model depends on unobserved latent variables.
- SMO: Sequential minimal optimization is an algorithm for solving the quadratic programming (QP) problem that arises during the training of support vector machines.

2) Key Ideas:

- Motivation: Their results suggest that some classification techniques tend to produce defect prediction models that outperform others on contrary to earlier research which stated the classification techniques didnt matter.
- Patterns: Logistic regression and linear regression, Multivariate Adaptive Regression Splines, Personalized Change Classification, and Logistic Model Trees. Ensemble methods that combine different machine learning techniques have also been explored. Lessmann et al. [14] conducted a study comparing the performance of 22 different classification techniques on the NASA corpus. Their results show that the performance of 17 of the 22 classification techniques are statistically indistinguishable from each other. Table I shows different kinds of classification techniques being studied.
- Assessment: To compare the performance of defect prediction models, they used the Area Under the receiver operating characteristic Curve (AUC), which plots the false positive rate against the true positive rate. They ran the Scott-Knott test to group classification techniques into statistically distinct ranks.

3) Conclusion and improvement: Ensemble techniques (i.e., RF+a base learner), decision trees (i.e., LMT), statistical techniques (i.e., Simple Logistic and Naive Bayes), neural networks (i.e., RBFs), and nearest neighbour (i.e., KNN) outperformed models trained using rule-based techniques (i.e., Ripper and Ridor), clustering techniques (i.e., K-means and EM) and SVM (i.e., SMO). There are scopes of improvement in following ways:

TABLE I
OVERVIEW OF THE STUDIED CLASSIFICATION TECHNIQUES.

Family	Technique	Abbreviation
Statistical Techniques	Naive Bayes	NB
	Simple Logistic	SL
Clustering Techniques	K-means	K-means
	Expectation Maximization	EM
Rule-Based Techniques	Repeated Incremental Pruning to Produce Error Reduction	Ripper
	Ripple Down Rules	Ridor
Neural Networks	Radial Basis Functions	RBFs
Nearest Neighbour	K-Nearest Neighbour	KNN
Support Vector Machines	Sequential Minimal Optimization	SMO
Decision Trees	J48	J48
	Logistic Model Tree using Logistic Regression	LMT
Ensemble Methods using LMT, NB, SL, SMO, and J48	Bagging	Bag+LMT, Bag+NB, Bag+SL, Bag+SMO and Bag+J48
	Adaboost	Ad+LMT, Ad+NB, Ad+SL, Ad+SMO and Ad+J48
	Rotation Forest	RF+LMT, RF+NB, RF+SL, RF+SMO, and RF+J48
	Random Subspace	Rsub+LMT, Rsub+NB, Rsub+SL, Rsub+SMO, and Rsub+J48

- They studied 2 datasets which have different predictor metrics, this difference might be the reason for their results. They should have unified the metrics properly and then have come out for a conclusion.
- This kind of study also gives rise to cross project defect prediction and unifying the predictor metrics.
- Since they tested on so many different techniques, they can now work on hyperparameter optimization of learners, to come up with other explanations.

D. *Automated Parameter Optimization of Classification Techniques for Defect Prediction Models [25]*

1) Keywords:

- Experimental Design: The design of experiments (DOE, DOX, or experimental design) is the design of

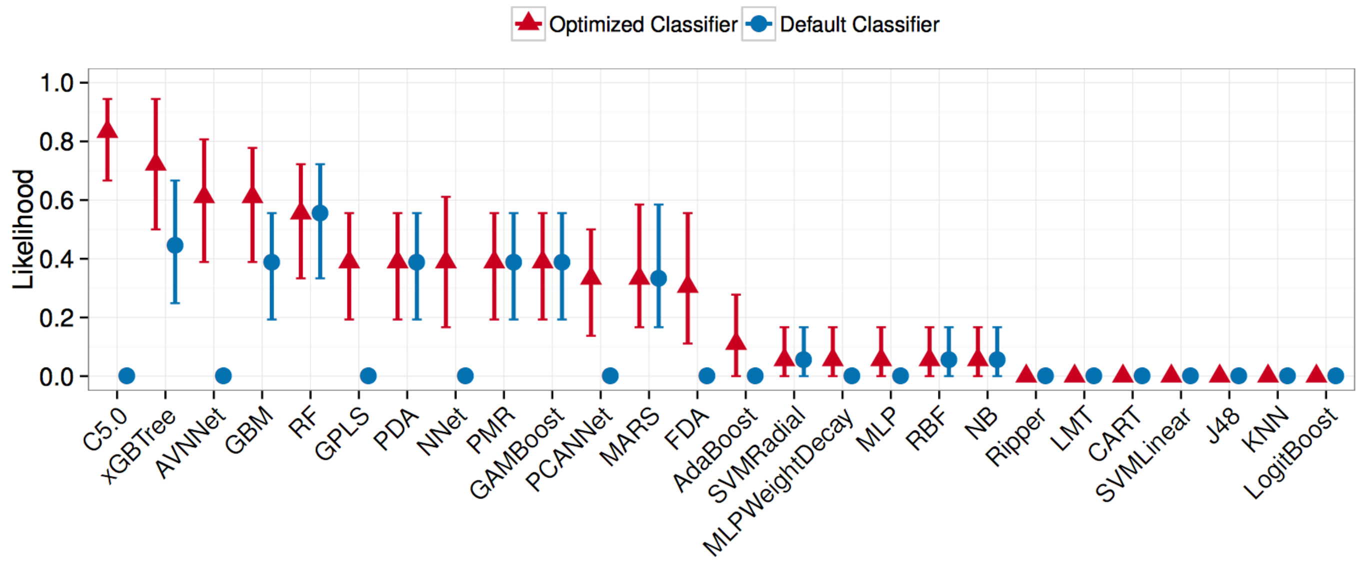

Fig. 1. The likelihood of each technique appearing in the top Scott-Knott rank. Circle dots and triangle dots indicate the median likelihood, while the error bars indicate the 95% confidence interval of the likelihood of the bootstrap analysis. A likelihood of 80% indicates that a classification technique appears at the top-rank for 80% of the studied datasets

any task that aims to describe or explain the variation of information under conditions that are hypothesized to reflect the variation.

- Classification Techniques: In machine learning and statistics, classification is the problem of identifying to which of a set of categories (sub-populations) a new observation belongs, on the basis of a training set of data containing observations (or instances) whose category membership is known. Eg. Naive Bayes classifier, Logistic regression etc.
- Parameter Optimization: In the context of machine learning, parameter optimization or model selection is the problem of choosing a set of parameters for a learning algorithm, usually with the goal of optimizing a measure of the algorithm's performance on an independent data set. Often cross-validation is used to estimate this generalization performance.

2) ***Key Ideas***:

- Motivation: Defect prediction models are classifiers that are trained to identify defect-prone software modules. Such classifiers have configurable parameters that control their characteristics (e.g., the number of trees in a random forest classifier). Recent studies show that these classifiers may underperform due to the use of suboptimal default parameter settings. However, it is impractical to assess all of the possible settings in the parameter spaces. This paper investigates the performance of defect prediction models where Caret [13] an automated parameter optimization technique has been applied.
- Related Work: Recent research [6], [1] has raised concerns about parameter settings of different data miners. Some other studies shows the effect of parameter settings of classification techniques when applied to defect prediction models. For example, Koru et al. [12] and Mende et al. [15] point out that selecting different parameter settings can impact the performance of defect models. Jiang et al. [10] and Tosun et al. [26] also point out that the default parameter settings of research toolkits (e.g., R, Weka, Scikit-learn, MATLAB) are suboptimal. Although prior work suggests that defect prediction models may underperform if they are trained using suboptimal parameter settings, parameters are often left at their default values. Recent research voices concerns about the stability of performance estimates that are obtained from classification techniques when applied to defect prediction models. For example, Menzies et al. [17] and Mittas et al. [18] argue that unstable classification techniques can make replication of defect prediction studies more difficult. Like any form of classifier optimization, automated parameter optimization may increase the risk of overfitting, i.e., producing a classifier that is too specialized for the data from which it was trained to apply to other datasets.
- Informative Visualization: The figure 1 summarises the results.

3) ***Conclusion and improvement***: C5.0 boosting tends to yield top-performing defect prediction models more frequently than the other studied classification techniques. Automated parameter optimization increases the likelihood of appearing in the top Scott-Knott rank by as much as 83%. Automated parameter optimization increases the likelihood of 11 of the studied 26 classification techniques by as much as 83% (i.e., C5.0 boosting). This suggests that automated parameter optimization can substantially shift the ranking of classification techniques. Since automated parameter optimization techniques like Caret yield substantially benefits in terms of performance improvement and stability, while incurring a manageable additional

computational cost, this paper suggests researchers should be included in future defect prediction studies. There are scopes of improvement in following ways:

- Since computational cost is high, we can go for much simple optimizers like differential evolution [24].

V. Discussion and Conclusion

After such an extensive study, we find some take away points. They are mentioned below.

- The combined or socio-technical models for any of the evaluation metrics can give a good prediction model.
- Using the CK and the OO metric sets together is preferable than using them in isolation.
- Random Forest, naive bayes, simple logistic, decision trees, and nearest neighbors perform better in defect prediction.
- Sometimes the data is imbalanced and we should balance the classes.
- Tuning the parameters of classification gives enormous difference in performance, so tuning can not be neglected.
- Cross project defect prediction should be our next concern.

References

[1] A. Agrawal, W. Fu, and T. Menzies. What is wrong with topic modeling?(and how to fix it using search-based se). *arXiv preprint arXiv:1608.08176*, 2016.

[2] C. Bird, N. Nagappan, H. Gall, B. Murphy, and P. Devanbu. Putting it all together: Using socio-technical networks to predict failures. In *2009 20th International Symposium on Software Reliability Engineering*, pages 109–119. IEEE, 2009.

[3] C. Catal and B. Diri. A systematic review of software fault prediction studies. *Expert systems with applications*, 36(4):7346–7354, 2009.

[4] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[5] M. D'Ambros, M. Lanza, and R. Robbes. An extensive comparison of bug prediction approaches. In *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*, pages 31–41. IEEE, 2010.

[6] W. Fu, T. Menzies, and X. Shen. Tuning for software analytics: Is it really necessary? *Information and Software Technology*, 76:135–146, 2016.

[7] B. Ghotra, S. McIntosh, and A. E. Hassan. Revisiting the impact of classification techniques on the performance of defect prediction models. In *Proceedings of the 37th International Conference on Software Engineering-Volume 1*, pages 789–800. IEEE Press, 2015.

[8] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell. A systematic literature review on fault prediction performance in software engineering. *IEEE Transactions on Software Engineering*, 38(6):1276–1304, 2012.

[9] A. E. Hassan. Predicting faults using the complexity of code changes. In *Proceedings of the 31st International Conference on Software Engineering*, pages 78–88. IEEE Computer Society, 2009.

[10] Y. Jiang, B. Cukic, and T. Menzies. Can data transformation help in the detection of fault-prone modules? In *Proceedings of the 2008 workshop on Defects in large software systems*, pages 16–20. ACM, 2008.

[11] S. Kim, T. Zimmermann, E. J. Whitehead Jr, and A. Zeller. Predicting faults from cached history. In *Proceedings of the 29th international conference on Software Engineering*, pages 489–498. IEEE Computer Society, 2007.

[12] A. G. Koru and H. Liu. An investigation of the effect of module size on defect prediction using static measures. In *ACM SIGSOFT Software Engineering Notes*, volume 30, pages 1–5. ACM, 2005.

[13] M. Kuhn. Caret: classification and regression training. *Astrophysics Source Code Library*, 1:05003, 2015.

[14] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch. Benchmarking classification models for software defect prediction: A proposed framework and novel findings. *IEEE Transactions on Software Engineering*, 34(4):485–496, 2008.

[15] T. Mende. Replication of defect prediction studies: problems, pitfalls and recommendations. In *Proceedings of the 6th International Conference on Predictive Models in Software Engineering*, page 5. ACM, 2010.

[16] T. Menzies, J. Greenwald, and A. Frank. Data mining static code attributes to learn defect predictors. *IEEE transactions on software engineering*, 33(1):2–13, 2007.

[17] T. Menzies and M. Shepperd. Special issue on repeatable results in software engineering prediction. *Empirical Software Engineering*, 17(1):1–17, 2012.

[18] N. Mittas and L. Angelis. Ranking and clustering software cost estimation models through a multiple comparisons algorithm. *IEEE Transactions on Software Engineering*, 39(4):537–551, 2013.

[19] N. Nagappan, T. Ball, and A. Zeller. Mining metrics to predict component failures. In *Proceedings of the 28th international conference on Software engineering*, pages 452–461. ACM, 2006.

[20] D. Radjenović, M. Heričko, R. Torkar, and A. Živkovič. Software fault prediction metrics: A systematic literature review. *Information and Software Technology*, 55(8):1397–1418, 2013.

[21] M. Shepperd, D. Bowes, and T. Hall. Researcher bias: The use of machine learning in software defect prediction. *IEEE Transactions on Software Engineering*, 40(6):603–616, 2014.

[22] Q. Song, Z. Jia, M. Shepperd, S. Ying, and J. Liu. A general software defect-proneness prediction framework. *IEEE Transactions on Software Engineering*, 37(3):356–370, 2011.

[23] Q. Song, M. Shepperd, M. Cartwright, and C. Mair. Software defect association mining and defect correction effort prediction. *IEEE Transactions on Software Engineering*, 32(2):69–82, 2006.

[24] R. Storn and K. Price. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11(4):341–359, 1997.

[25] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto. Automated parameter optimization of classification techniques for defect prediction models. In *Proceedings of the 38th International Conference on Software Engineering*, pages 321–332. ACM, 2016.

[26] A. Tosun and A. Bener. Reducing false alarms in software defect prediction by decision threshold optimization. In *Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement*, pages 477–480. IEEE Computer Society, 2009.

[27] T. Zimmermann and N. Nagappan. Predicting defects using network analysis on dependency graphs. In *Proceedings of the 30th international conference on Software engineering*, pages 531–540. ACM, 2008.