The University of Saskatchewan
Department of Computer Science

Technical Report #2013-02

UNIVERSITY OF SASKATCHEWAN

Adaptive Bug Classification for CVE List using Bayesian Probabilistic Approach

Mohammad Masudur Rahman, Shamima Yeasmin
Computer Science, University of Saskatchewan, Canada
{mor543, shy942}@mail.usask.ca

***Abstract*—Software bug classification is a precondition for bug fixation and it plays a vital role in software maintenance. It is found that bug fixation often takes long due to the distribution of misclassified or non-classified bugs by the triager among the developers. In this paper, we propose an adaptive bug classification approach on *CVE* dataset that involves two Bayesian classifiers such as Naive Bayes and Bayes net, and takes adaptive decision for classification. Naive Bayes is a classification algorithm which adopts a naive approach regarding class conditional distribution during classification and assumes that all the features of a sample are conditionally independent given the class label. Bayes net is a graphical representation of a set of random variables and their conditional dependencies via a directed acyclic graph. It can be used for knowledge representation as well as classification. Exploiting the domain knowledge about the bug class, we conduct the experiments from two different view points - group-based approach and general approach. In case of group-based approach, both classifiers are learned using the bug group-specific samples and selected features from five groups. In case of general approach, 28,266 bug samples and 64 bug features are considered. Experiments show that Bayes net classifier has more potential than Naive Bayes for classification with *CVE* dataset and therefore it is preferable. However, it is also found that Naive Bayes classifier performs fairly well in *CVE* bug classification due to its simplistic concept and less number of parameters.**

***Index Terms*—Software bug classification; Bayes net; CVE; Conditional independence; ExTax.**

I. Introduction

Software bug classification is a precondition for bug fixation and it plays a vital role in software maintenance. Once a bug is reported, the triager [1] tries to assign it to the available and interested developer for fixation. However, it is found that bug fixation often takes long specially when the bug assignment is not appropriate for the developer. Each developer has own set of skills. So, when the triager assigns a bug that requires skills beyond the skill matrix of the developer for fixation, then the bug either gets an improper fixation or a delayed fixation by the developer. This often results into a reopening of the bug which is a costly operation for software maintenance (Fig. 1: Software Bug fixation cycle). This is the case when triager distributes misclassified or non-classified bugs. So, bug classification is an essential task and it can help the triager to correctly assign the bugs to the appropriate developers according to their expertise and interests, and thus, it can reduce the bug fixation time and cost.

Most of the existing approaches for software bug classification are necessarily subjective, manual or semi-automated [7]. Billah and Roy [8] propose a framework, $ExTax$, for extensible bug classification on CVE [4] dataset which is a collection of common security and vulnerability issues of different security tools, software repositories and databases from all over the world. They propose 22 bug classes under 5 groups considering 64 bug features. They extract features manually and classify 25,357 bugs in a semi-automated way; however, they do not apply any machine learning technique in their approach. Actually, to the best of our knowledge, no machine learning technique is used against CVE dataset for bug classification which motivated us to investigate the performance of a machine learning technique such as Bayesian Probabilistic approach for bug classification in this dataset.

In this paper, we propose an adaptive bug classification approach on CVE dataset that involves two Bayesian classifiers (e.g., Naive Bayes, Bayes net) and takes adaptive decision for classification. We also conduct a comparative study between those two classifiers. While Naive Bayes assumes conditional independence among the bug features, Bayes net exploits the dependency relationships among them for bug classification. They also help us to perform insightful analysis of the characteristics of dataset and to explore significant facts about the problem domain . Throughout this research, we try to answer three research questions related to bug features and their implications, the effectiveness of our approach etc. They are- (1) Are the bug features conditionally independent of each other given the bug classes? If not, what are those dependencies and how are they influencing in the classification? (2) In case of our dataset, the bug features are manually extracted and it contains missing feature values for most of the samples; however, we can not discard those samples for obvious reasons. So, which expected values of those features do maximize the bug classification accuracy? (3) How much effective is the Bayesian probabilistic approach for CVE bug classification?

We reuse the proposed classification scheme and the bug features extracted by Billah and Roy [8] in our experiment. We conduct the experiments from two different view points. In the first case, the proposed approach uses domain knowledge for feature selection for each group and applies both Naive Bayes and Bayes net for classification. It learns the Bayes net structure using different local score based search algorithms, and estimates the parameters for each group individually. In this case, we got 81.38% and 78.15% average classification accuracy for each group for Naive Bayes and Bayes net clas-

[1]The person who assigns bug to the developers

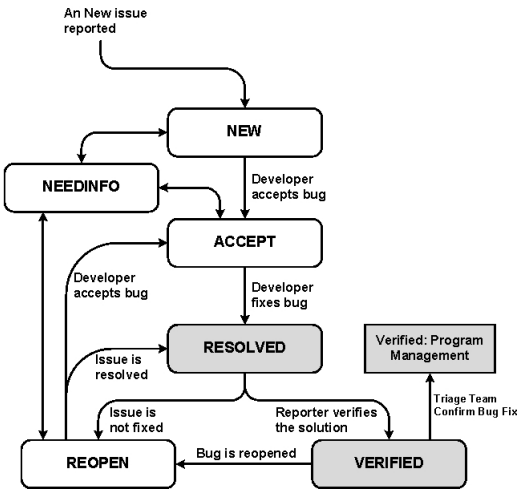

Fig. 1. Software Bug fixation cycle

sifier respectively. In the second case, our approach considers all 64 features associated with different groups and learns a single model for Naive Bayes classifier. It also learns a single Bayes net comprising 65 nodes. The idea is to disregard the feature selection concept, and explore and exploit the hidden dependency relationships among all the features for bug classification. In this case, we got 80.64% and 81.50% classification accuracy for Naive Bayes and Bayes net respectively. Moreover, we delved deeper and found that the assumption of conditional independence among the bug features for Naive Bayes classifier works fairly well against it low cost training, and the feature selection concepts can significantly improve the classification accuracy through imposing a few constraints on the features. We also noted that the learned Bayes nets provide better classification accuracy than naive Bayes for increased number of samples with a compromise in training cost; moreover, they also help us to explore and reason about different facts related to the bug features and the classes.

The rest of the paper is structured as follows. Section II focuses on the brief literature review, Section III describes details of the dataset we used and Section IV discusses about our proposed bug classification approach. Section V elicits the experimental results and Section VI concentrates on the overall results and the implications of our findings. Section VII discusses about our future plans with this work and finally, Section VIII summarizes the whole paper.

II. Background

Existing studies on bug classification can be divided into two broad categories - subjective approach which relies on human discretion or judgement and objective approach which emphasizes on methodology generalization through some fixed rules. Seaman et al. [14] aggregate historical data to model defect taxonomy and propose a subjective and extensible approach for bug classification. However, the approach is completely manual.

Nakamura et al. [12] present a methodology of defect analysis that involves investigating the defects in the source code level. Knuth [11] proposes an approach for TEX error classification. However, both of the approaches are highly subjective and require expert knowledge about defect analysis.

Another subjective taxonomy is Unix Security Taxonomy by Aslam [6]. He enlists the defects leading to security problems in UNIX operating system in a hierarchical manner. In hierarchical taxonomy, defect classes are organized in several levels and any class generalizes a set of classes from the level below it.

To characterize software defect types, Chillarege et al. [9] propose a defect classification scheme called *Orthogonal Defect Classification(ODC)* which captures semantic information from defect description. The scheme assigns attributes to a defect and represents the defect in a n-axis Cartesian coordinate system where each attribute refers to an axis. $ExTax$ framework by Billah and Roy [8] exploits the *ODC* idea of objective interpretation and defect representation as a point in n-dimensional system. It also borrows the idea of hierarchical representation of bug taxonomies [6] and extensible bug taxonomy [14] which make it a framework considering both subjective and objective aspects during bug classification. Our research is influenced by this framework as we reuse its subjective interpretation of bug groups, classes and attributes. However, our approach contrasts with $ExTax$ by applying machine learning approach for classification. It also considers two view points for classification (e.g., group-based and general) exploiting the domain knowledge about bug classes and features.

In a recent study, Neelofar et al. [13] use multinomial Naive Bayes classifier on 29,000 BugZilla samples from Eclipse and Mozilla and extract features using *Chi Square* and *TFIDF* algorithms. Their approach shows an average classification accuracy of 83%. In our research, we use both Naive Bayes and Bayes net for bug classification on a different dataset and more importantly we manipulate the causal or dependence relationships among the bug features for better classification and insightful analysis of the dataset.

III. Dataset

In our research, we use *CVE List* [4] as the dataset which is a collection of 61,894 reported incidents about security vulnerabilities and exposures of different security tools, software repository, database etc from all over the world. However, we use a subset of 25,350 defects for the experiment which are manually classified by Billah and Roy [8]. It needs to be mentioned that they classify several samples under multiple classes. Let us consider that a sample is classified under class *C1* and *C2*. We consider it two separate samples under class *C1* and *C2* and thus, we get a set of 28,266 samples. The idea is to investigate the applicability and the performance of the machine learning approach such as Bayesian probabilistic approach on the same dataset for bug classification. In the following sections, we discuss different important characteristics of our dataset.

TABLE I
BUG GROUPS AND CLASSES

Group (Level 1)	#Class	Class (Level 2)	
Computation (C)	3	Value Representation Defect (C.1) Undefined Outcome (C.3)	Value Offset Defect (C.2)
Logic (L)	8	Improper Checks (L.1) Wrong Operation (L.3) Performance Issues (L.5) Control Flow Error (L.7)	Improper Terminal Conditions (L.2) Flaws in Algorithm (L.4) Improper Exception Handling (L.6) Design Non-conformance (L.8)
Memory (M)	4	Invalid Memory Reference(M.1) Memory Leaks (M.3)	Improper Deallocation (M.2) Over/Underflow (M.4)
Data, Interface and Input/Output (D	5	Interface Mismatch (D.1) Improper Input Validation (D.3) Improper Abstraction (D.5)	Data Mismatch (D.2) Missing or Extra Input (D.4)
Synchronization (S)	2	Prohibitive States (S.1)	Improper Sequencing (S.2)

A. Bug Classes and Class hierarchy

Billah and Roy [8] propose two level classification for *CVE* dataset where level one ensures the subjectivity[2] and level two ensures objectivity[3] of the bug classification simultaneously. In our research, we also consider two level classification where level one class (group) information for a bug comes in the form of domain knowledge (a.k.a., subjective interpretation of the bug) and level two class information comes from machine learning classifiers. Our dataset has five groups or level one classes- Computation (C), Logic (L), Data interface and Input/Output (D), Memory (M) and Synchronization (S). Each bug group contains several bug classes which interpret the group from objective point of view. The dataset contains 22 classes in total under five groups. Table I shows the bug groups and their corresponding bug classes of our dataset. For example, Logic (L) is a bug group, and *Flaws in Algorithm (L.4)* is a bug class under the group and it is interpreted by some fixed rules.

B. Bug Features

Each bug class of the dataset is characterized by several bug features and we consider 64 features in total for 22 bug classes. Table VI in Appendix shows the bug features. From Table VI, we can see that each bug group contains fixed number of features under its classes. The classes under Logic (L) and Synchronization (S) groups contain the maximum and minimum number of features respectively. We also note that a bug can be classified under multiple bug classes, but not under multiple groups; that means, during classification, bug groups (level one) are mutually exclusive, but, bug classes (level two) are not.

Generally, bug description is written in natural language text and bug feature extraction is a non-trivial task which is often done manually taking help from domain experts. In this research, we use the extracted features by Billah and Roy [8], and each feature value represents the boolean response of a particular incident related to the bug. For example, the classes under Logic (L) group contain a bug feature -*Missed to save a value* which has a value of *yes*(1) or *no*(0). Thus, all the feature values for our dataset are drawn from binary distribution.

[2]subjective interpretation of bug
[3]a set of formal rules to characterize the bug

TABLE II
BUG SAMPLES STATISTICS

Bug Group	#TS[1]	# SCFV[2]	#SCMV[3]	PSCMV[4]
Computation (C)	89	3	86	96.62%
Logic (L)	3,499	217	3,282	93.80%
Memory (M)	4,704	831	3873	82.33%
Data, Interface & I/O (D)	18,292	6,722	11,570	63.25%
Synchronization (S)	1,682	185	1497	89.00%
All Groups	28,266	7,958	20,308	71.84%

[1] Total samples
[2] Samples containing all feature values
[3] Sample containing missing feature values
[4] Percentage of samples containing missing feature values

C. Bug Sample Statistics

Table II shows detailed statistics about the bug samples, missing feature values etc from different bug groups. From Table II, we can see that the dataset contains 18,292 samples from *Data, Interface and I/O (D)* group whereas only 89 samples from *Computation (C)* group. So, there is quite an unequal distribution of samples. As the bug features are manually extracted, a major portion of the samples contain missing values. We find that about 71.84% of the total samples have one or more features containing missing values. We try to analyze a few bug sample descriptions against the missing feature values in the dataset and find that it is a non-trivial task to populate all feature values from the bug report. Thus, missing feature value is a common issue for manual feature selection. However, we deal with it satisfactorily which we discuss in Section IV-E.

IV. ADAPTIVE BUG CLASSIFICATION FOR CVE LIST USING BAYESIAN PROBABILISTIC APPROACH

Our proposed approach can take adaptive decision for classification based on available information about the bug samples. Here, we use two Bayesian classifiers- Naive Bayes and Bayes net and consider two view points for classification- group based approach and general approach. Fig 2 shows different steps and modules involved into our proposed approach. In the next few sections, we describe them in detail.

A. Naive Bayes Classifier

Naive Bayes classifier is a classification algorithm which adopts a naive approach regarding class conditional distribution during classification and it assumes that all the features of a bug sample are conditionally independent given the class

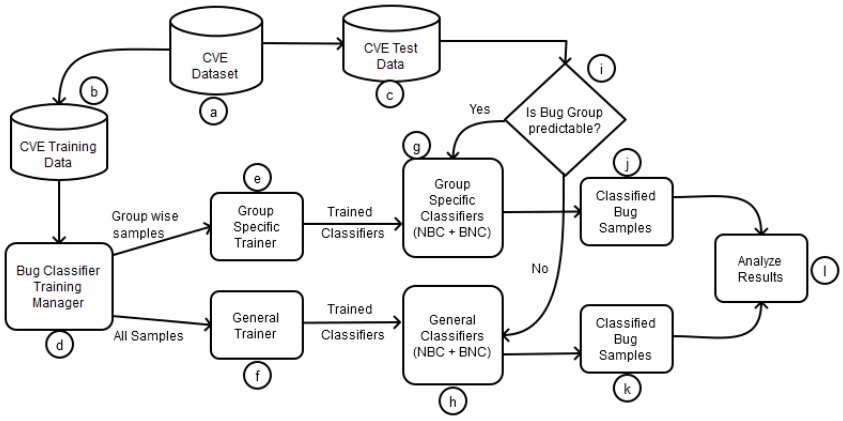

Fig. 2. Schematic Diagram of Proposed Approach for Bug Classification

label. However, if the assumption is not correct, it still works well. Because it has a limited number of parameters to estimate which make it immune to overfitting. There are two steps in the classification with Naive Bayes classifier-model fitting and using the model for posterior prediction.

1) Model Fitting: The first step with Naive Bayes classification is to develop the model for the classifier. It involves estimating certain parameters like maximum likelihood (ML) or maximum aposteriori (MAP). If we consider a training set containing *C* classes with *D* features, then the log-likelihood of the training data given the parameters θ can be calculated using the following equation.

$$\log p(Data|\theta) = \sum_{c=1}^{C} N_c \log \pi_c + \sum_{j=1}^{D} \sum_{c=1}^{C} \sum_{i:y_i=c} p(x_{ij}|\theta_{jc}) \quad (1)$$

Here, the expression can be decomposed into a series of terms concerning π, the class prior and *DC* terms containing the θ_{jc}'s, the likelihood of class features. Thus, we can estimate the maximum likelihood of class prior, $\hat{\pi}_c$, as the following:

$$\hat{\pi}_c = \frac{N_c}{N} \quad (2)$$

where, $N_c = \sum_i I(y_i = c)$ is the number of samples in class c. We also can estimate the ML of likelihood for each feature considering a distribution; in our case, all features are binary and therefore we assume $x_j|y = c \sim Ber(\theta_{jc})$. Thus, we get the MLE for likelihood of each feature using the following equation.

$$\hat{\theta}_{jc} = \frac{N_{jc}}{N_c} \quad (3)$$

Calculation of MLE parameters for class prior and likelihood of data (features) are quite simplistic and involves empirical counting. However, the model comprises of MLEs is often prone to overfitting and may perform badly when it observes the testing samples that are unlikely. Therefore, we need to perform smoothing operation on the empirical counts. To perform smoothing and make the model immune to overfitting, we consider a conjugate $Dir(\alpha)$ prior for π and a conjugate $Beta(\beta_0, \beta_1)$ prior for θ_{jc}. They add pseudo parameters to the empirical counts due to their conjugacy attributes. Thus, we can estimate the MAP parameters using the following equations.

$$\hat{\pi}_c = \frac{N_c + \alpha_c}{N + \alpha_0} \quad (4)$$

$$\hat{\theta}_{jc} = \frac{N_{jc} + \beta_1}{N + \beta_0 + \beta_1} \quad (5)$$

Here, $\alpha_0, \alpha_c, \beta_0, \beta_1$ are pseudo parameters. As a simple guess, we consider uniform priors for all classes and features and therefore, $\alpha_0 = \alpha_c = \beta_0 = \beta_1 = 1$.

2) Prediction with the model: Once the model is developed, we use it for posterior prediction using the following equation where the point estimates such as MLE or MAP are plugged in.

$$p(y = c|x, D) \propto \hat{\pi}_c \prod_{j=1}^{D} (\hat{\theta}_{jc})^{I(x_j=1)} (1 - \hat{\theta}_{jc})^{I(x_j=0)} \quad (6)$$

Here $\hat{\pi}_c, \hat{\theta}_{jc}$ are the point estimates and will be replaced by MLE or MAP estimation for classification. Basically, we determine the $\hat{c}^{MAP}$ using Equation 9 to determine the class of a bug sample.

B. Bayes Net Classifier

Bayes net is a graphical representation of a set of random variables and their conditional dependencies via a directed acyclic graph. Here, the random variables are denoted by nodes and their conditional dependencies are denoted by connecting edges. The edges can be directed or undirected. If there is no edge between two nodes, that means they are conditionally independent whereas if there is an edge, that means they are conditionally dependent. For example, Fig. 3 shows a Bayes net where the attributes (A_1, A_2, A_3) are conditionally dependent on class node C. Bayes net is often termed as *belief networks* or *causal networks*.

As Bayes net structure encodes the important probabilistic relationships among the random variables, its structure can be learned and parameters can be estimated to use for inferencing about a set of variables in the net when another set of variables

are given. Thus, it can be used for classification. Using Bayes net as a classifier involves three steps - learning structure, learning parameters and prediction (inference) with Bayes net.

1) Learning Bayes net structure: In our approach, we consider Bayes net as a directed acyclic graph and we use several local score-based search algorithms for learning the structure such as *K2, Hill-climbing, Tree-augmented Naive Bayes (TAN), Simulated annealing* etc. The idea behind the multiple search algorithms is to determine which algorithm provides the structure that best reflects the data characteristics. Here, we describe those algorithms in brief.

a) K2: It is a greedy search algorithm used for learning when the total ordering such as parents, children are known in advance. These ordering can be used to reduce the search space. The algorithm starts learning with a list of nodes that have no parents. It then adds the parent nodes that increase the total score of the network. When any parent addition does not increase the total score, it stops adding the parent nodes [2] and finalizes the structure.

b) Hill-climbing: It is a greedy algorithm that adds a neighbour node to the structure that improves the total score of the network. However, this greedy behaviour often leads to several problems like *local maxima, plateau, ridges* etc and the model results into a non-optimal model [2].

c) Simulated annealing: This algorithms solves the *local maxima* problem of *hill-climbing* algorithm. It does not move to the neighbour structure containing highest score always, rather it follows a stochastic way for switching. It switches to a neighbour structure containing the highest score with a probability of p and performs a random walk to another node with a probability of $1-p$ [3]. This stochastic behaviour helps the model to escape the local maximum and find a global maximum.

d) Tree augmented search (TAN): . The algorithm starts searching with an initial Bayes net structure of Naive Bayes classifier. It forms a maximum weighted spanning tree by adding edges among the attribute nodes and the tree is formed using *Chow and Liu* algorithm [10].

For all search algorithms, we use well known *Bayes score* as the metric for scoring function. Thus, each algorithm selects the best network, G, based on its posterior probability given the training data as stated by the following equation.

$$p(G^k|D) = p(D, G^k)/p(D) \tag{7}$$

Here, $P(D)$ is the normalization constant which does not depend on the Bayes net structure. So, the relative posterior probability, $p(D, G^k) = p(D|G^k)p(G^k)$ is often used for model selection where $p(D|G^k)$ is the marginal likelihood of the network and $p(G^k)$ is the prior assumption about the network. We consider a uniform prior for all networks.

2) Estimating Bayes net parameters: Once the Bayes net structure is learned and selected, we construct the *conditional probability table* for each node in the graph. We manipulate the graph structure for the computation. Bayes net follows the Markov property; that means each node is conditionally independent of its non-descendants given the parents. Thus the joint probability distribution of the network can be obtained using the following equation.

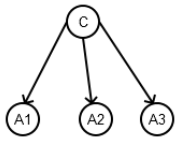

Fig. 3. Bayes Net Example

$$p(X_1, X_2, ...X_n) = \prod_i p(X_i|pa(X_i)) \tag{8}$$

Here, $pa(X_i)$ represents the parent of node of X_i. We use the point estimates such as MAP and ML following equation 5 and 4 to calculate the conditional probability for the nodes in the graph.

3) Bayes net for classification: After the Bayes net is selected and the network parameters are learned, it can be used for classification. The idea is to exploit the joint probability distribution of Bayes net (which can be found using Equation 8) to calculate the posterior prediction of a class for a given sample using the following equation.

$$\hat{c}^{MAP} = \underset{c}{argmax}[p(Data|\theta)p(\theta)] \tag{9}$$

Here, θ represents the learned parameters from Bayes net and $Data$ represents the sample features. Thus, Bayes net is basically a Bayesian classifier with reduced computation effort and better support against overfitting.

C. Group Based Approach for Classification

Subjective interpretation of the bug is often helpful for bug classification or bug fixation [8]. In our classification approach, we are interested to manipulate it during classification. We consider level one classes or groups as the subjective views of the original bug classes. The idea is, if the group of a bug sample can be predicted from its title or description text, then it should be classified using group-based classification approach, otherwise it will be classified using general approach. The triager will be responsible to derive the group related information of a bug sample based on the bug sample content and his/her domain expertise. In group based approach, we consider five groups - Computation (C), Logic (L), Memory (M), Data, Interface and I/O (D) and Synchronization (S), and develop five group-based classifiers (Fig .2-(g)). Each group-based classifier contains one naive Bayes and one Bayes net classifier and we select a subset of 64 bug features for each group. We analyze the complete dataset and perform attribute selection based on our observation and the feedback of the domain experts for each group.

Table III shows the detailed statistics about the bug classes, features and parameters for the group-based classifiers. Here, we can see that the Naive Bayes classifier for Memory (M) group needs to learn 52 model parameters whereas a Bayes net of the same group needs to learn 13 CPTs. Fig. 4 and Fig. 5 show the learned Bayes nets for Computation (C) and

TABLE III
GROUP BASED CLASSIFICATION

Bug Group	#BC[1]	# BF[2]	#PNB[3]	#CPTBN[4]
Computation (C)	3	7	24	8
Logic (L)	8	28	232	29
Memory (M)	4	12	52	13
Data, Interface & I/O (D)	5	12	65	13
Synchronization (S)	2	5	12	6

[1] Bug classes under the group
[2] Bug features selected for the group
[3] Parameters to be learned for Naive Bayes classifier
[4] CPTs to be calculated for selected Bayes net.

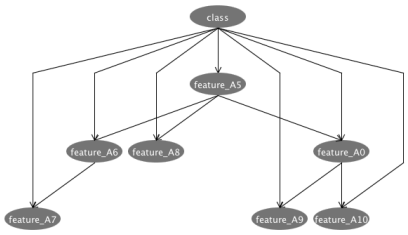

Fig. 4. Computation (C) Group Bayes net

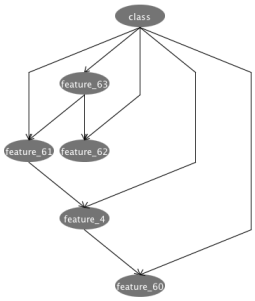

Fig. 5. Synchronization (S) Group Bayes net

Synchronization (S) groups that are learned by *TAN* search algorithms and we used for group-based bug classification.

D. General Approach for Classification

General approach for classification is used when the group specific information about the bug sample is not available. In case of general-approach classifier, we consider all 64 bug features and 22 classes (level two) proposed by Billah and Roy [8] and develop a naive Bayes classifier and a Bayes net classifier (Fig. 2-(h)). For naive Bayes classifier, the model learns 1430 model parameters whereas the Bayes net learns 65 conditional probability tables.

E. Handling Missing Values

The dataset has about 71.84% samples that contain one or more missing feature values. We use *Weka* to perform training and testing with our classifiers applying 10-fold cross validation. *Weka* replaces the numerical missing values with the global means; so, in both naive Bayes and Bayes net, the missing feature values are *replaced* by the global means for the training and classification purpose.

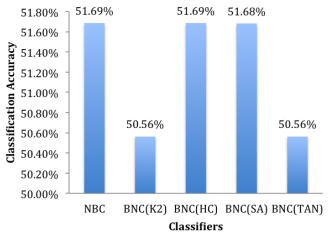

Fig. 6. Computation (C) Group Classification Accuracy

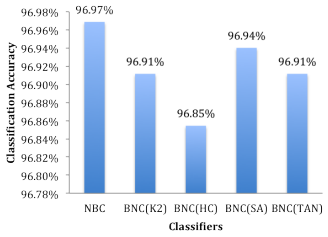

Fig. 7. Logic (L) Group Classification Accuracy

V. EXPERIMENTAL RESULTS

In this section, we discuss about the experimental results obtained from the proposed classification approach (e.g., group-based and general classification approach). Since the goal of this research is to conduct a comparative analysis between Naive Bayes and Bayes net classifiers for bug classification, our attempt is successful in overall and we find that Bayes net is preferable to Naive Bayes for our purpose. However, we analyze the classifier performance in terms of the following metrics.

- Classification accuracy (Recall)
- Cohen's Kappa statistics
- Area under ROC curve (AUC)
- Sample size vs accuracy curve
- Misclassification cost
- Bayes net Knowledge Exploration

A. Classification Accuracy (Recall)

For both approaches of bug classification, we calculated the classification accuracy for Naive Bayes and Bayes net classifiers. For Bayes net, we used four search algorithms for structure learning -K2, Hill Climbing (HC), Simulated Annealing (SA) and Tree Augmented Naive Bayes (TAN). Fig. 6, 7, 8, 9 and 10 show the classification accuracy of Bayes nets with different search algorithms and Naive Bayes classifiers for group-based approach. Here, we can see that Naive Bayes and Bayes net perform comparatively the same except Data, Interface and I/O (D) group. Here, Naive Bayes outperforms the Bayes net classifier. The maximum classification accuracy is found for Logic (L) group whereas the lowest accuracy is found for Computation (C) group.

Fig. 11 shows the classification accuracy for Naive Bayes and Bayes net classifiers for general approach based bug

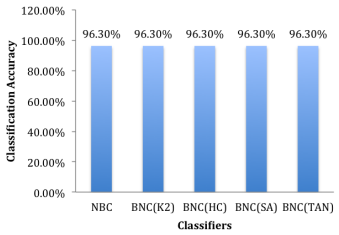

Fig. 8. Memory (M) Group Classification Accuracy

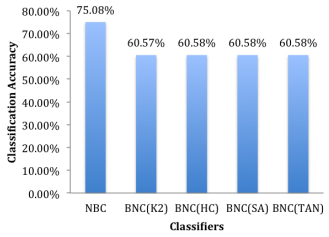

Fig. 9. Data, Interface, I/O (D) Group Classification Accuracy

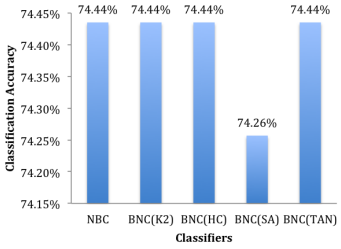

Fig. 10. Synchronization (S) Group Classification Accuracy

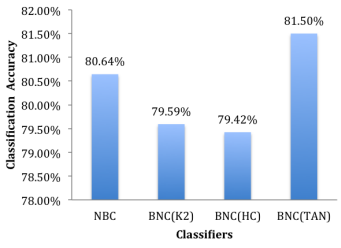

Fig. 11. All Samples Classification Accuracy

classification. The approach considers all 28,266 samples disregarding the group specific domain knowledge. Here, Bayes net performs classification with 79.59%, 79.42% and 81.50% accuracy with K2, Hill-climbing (HC) and Tree Augmented Naive Bayes (TAN) search algorithms respectively and Naive Bayes classifier has an accuracy of 80.64%. Obviously, this is not a great improvement for Bayes net, but it shows that Bayes net is likely to perform better for large sample set which we observed also from another test involving different sample sizes.

TABLE IV
KAPPA STATISTICS (GENERAL APPROACH)

Classifier (All samples)	Kappa (κ)
Bayes net (K2)	0.7369
Bayes net (HC)	0.7349
Bayes net (TAN)	0.7592
Naive Bayes	0.7495

TABLE V
KAPPA STATISTICS (GROUP-BASED APPROACH)

Classifier	Kappa (κ)(C)	Kappa (κ)(L)
Bayes net (K2)	-0.0103	0.0655
Bayes net (HC)	0.0000	0.0000
Bayes net (SA)	0.0000	0.0000
Bayes net (TAN)	-0.0137	0.0663
Naive Bayes	0.0000	0.0672

B. Cohen's Kappa Statistics

Cohen's Kappa is an important statistical measurement that represents how much the automated classification algorithm agrees with the actual classes of the bug samples. This is a value between 0 to 1 where 1 represents the complete agreement and 0 represents the total disagreement [5]. Table IV shows the *Kappa* statistic for Bayes net with different search algorithms and Naive Bayes classifiers. Here, the κ values are not significantly different and fall within the range of 0.61 to 0.80, that means, they all agree with the original bug classes strongly [5]. Table V shows the κ values for the classifiers of two groups. For convenience, we consider two extreme cases - Computation (C) and Logic (L).

From Table V, we can see that both Naive Bayes and Bayes net classifiers have very low κ values, that means, they barely agree with the original classes of the bug samples. Logic (L) group has a maximum classification accuracy of 96.97%, but has a κ value of 0.0672 which indicates poor agreement [5]. This may happen due to the limited number of bug samples under classification in the group-based approach (e.g., C has 89, L has 3499 samples), because for another group, D, we obtained $\kappa = 0.5929$.

C. ROC Curve Area

The Area under *ROC* curve (AUC) is also considered as an important metric for classification performance [1]. We calculated the area under *ROC* curve for each bug class and plot against different classifiers. Fig.12 shows the *AUC* plots for Bayes net and Naive Bayes classifiers from general approach where 22 classes are considered. Here, we can see that *AUC* values for the two classifiers are almost the same except two classes - *L8* with ID 11 and *M2* with ID 13. However, Bayes net has an average *AUC* value of 0.955 and Naives Bayes has 0.947; that means, both of the classifiers are excellent for testing, but Bayes net performs a little better than Naive Bayes [1].

D. Sample Size vs Accuracy Curve

We considered seven different sample sizes under general approach and calculated the classification accuracy measures

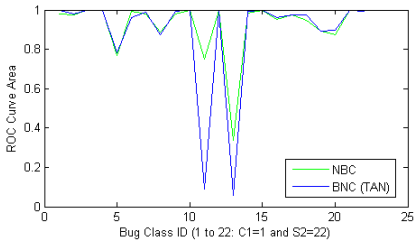

Fig. 12. ROC Area Statistics (All Samples)

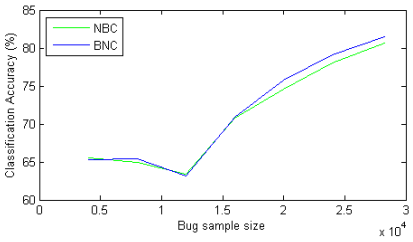

Fig. 13. Sample Size vs Classification Accuracy Curve

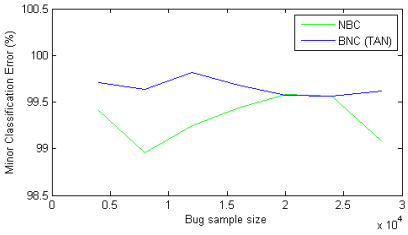

Fig. 14. Sample Size vs Minor Misclassification Error

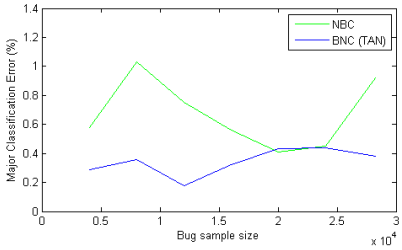

Fig. 15. Sample Size vs Major Misclassification Error

for both Bayes net and Naive Bayes classifiers. We found that with the addition of new samples, the classification accuracy increases slowly. However, we noted a decrease in the accuracy with 8000 and 12,000 sample sizes for both of the classifiers; this may happen due to the unequal distribution of the bug samples containing missing values in the dataset. Because, the sample set with 4000 samples contains no missing values and the set with 16000 samples contains 50% missing values. On the other hand, the target sample sets (i.e., 8000, 12000) contain about 42 ($< 1\%$) and 4000 (33.68%) samples having missing feature values respectively. However, with the increase of sample size, this effect reduces for both classifiers. Fig 13 shows the classification accuracy against different sample sizes for Naive Bayes and Bayes net classifiers.

From Fig. 13, we can see that the classification accuracy for both classifiers increases with the addition of new bug samples; however, it increases for Bayes net a bit faster than Naive Bayes. Thus, the we got a final classification accuracy of 81.50% for Bayes net and 80.46% for Naive Bayes classifier.

E. Misclassification Cost

We calculated the cost of misclassification by the classifiers. In this research, we considered two levels for classification - groups (a.k.a subjective interpretation of bug class) and original classes (level two). The motivation behind bug classification is to aid the triagers and the bug fixers with sufficient information about the type or class of the bug samples. However, the classifier may fail to return the correct class for a sample; but, the costs of all misclassification errors are not same. Let us consider the original class of a bug sample is $C1$, but the classifier returns $C3$ which is another class within the same bug group. We considered this type of misclassification as a *minor* error; however, if it returns $L4$ which is a class from another group, then the cost is significant and we considered it as *major* error. Because, in the former case, bug assignment decision may not be affected much, but, the later case may result into a potentially wrong assignment. These type of errors occurred in case of general approach for classification and we calculated those errors to contrast between Naive Bayes and Bayes net classifiers in terms of error cost.

We found from the previous statistics about classifier performances that Naive Bayes and Bayes net perform almost equally. However, cost and type of misclassification can add a new dimension to their evaluation metrics set. From Fig 14 and Fig. 15, we can see that Bayes net is relatively less prone to major misclassification than Naive Bayes, which denotes that Bayes net can be a good choice over Naive Bayes for classification.

F. Bayes net Knowledge Exploration

Besides classification Bayes net is a useful tool for knowledge representation and mining new information about the class and attributes involved into it. For both approaches of bug classification, we used Bayes net and learned the structure of Bayes net using different search algorithms. We also explored different information related to the bug features such as their conditional dependence, priority, distribution etc. For example, in the Bayes net structure involving 64 bug features, we found three clusters centering around three bug attributes - A6 (Expanded values), A8 (Specific offset of values) and A59 (Failed to provide access to a member). Fig. 16 shows a portion of our learned Bayes net.

From Fig 16, we can see that the bug feature *A8* is a parent node to multiple feature nodes such as *A28, A33, A34* etc, that means, those nodes are conditionally dependent on the parent node *A8*. However, according to the domain knowledge (i.e., subjective interpretation of bug), there is a little chance of having such conditional dependence, because *A8* and *A28, A33 and A34* are from different bug groups. But, from the training data set, the search algorithms developed that structure

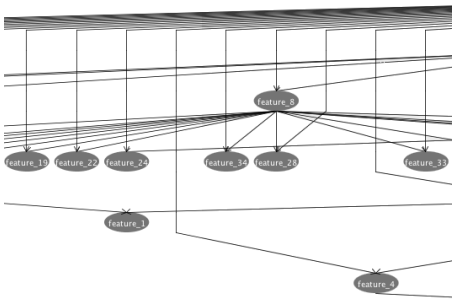

Fig. 16. Bayes net Knowledge Exploration

with *Bayes score* and therefore with maximum posterior probabilities. So, this new information is not ignorable and it can be considered as a new piece of fact which can encourage to rethink about the subjective interpretation of bugs classes. Thus, Bayes net can help in exploring valuable information that can be used for better classification and inferencing.

VI. Result Discussions & Implications

In this section, we discuss about the overall results of our proposed classification approach and the implications of the findings.

A. Overall Classification Results

In the previous section, we discuss about different metrics related to classifier performance. In case of group-based approach, Naive Bayes classifier performs better than or similar to Bayes net classifier for all groups except Synchronization (S). In our dataset, the bug samples are not equally distributed and therefore different groups contain different number of samples. For example, Computation (C) group contain only 89 bug samples out of 28,266 samples whereas Data, Interface and I/O (D) group contains 18,292 samples. These sample size and the existence of missing feature values influence the classification performance of the classifiers. We noted different classification accuracy for different groups. For example, both classifiers associated to Computation (C) group have the lowest accuracy whereas Logic (L) group classifiers have the highest classification accuracy. One possible reason behind the highest accuracy for Logic (L) group may be the largest feature set which is 28, that means, classifier can detect a sample efficiently with more features specified.

In case of general approach, when we disregard the subjective interpretation of bug classes and consider all 64 features for every bug class, Bayes net performs slightly better (i.e., classification accuracy 81.50%) than Naive Bayes classifier (i.e., classification accuracy 80.64%). That means, increased number of features and samples help the Bayes net to capture the bug feature dependencies more efficiently which results into a structure that fits with the semantics of dataset more accurately. However, it is interesting to note that the independence assumption among bug features (i.e., Naive Bayes) also works fairly well.

Besides classification accuracy, κ statistic is also an important metric for classifier performance [5]. For general approach, both Naive Bayes and Bayes net show a strong agreement with the original bug classes (i.e., $\kappa_{NBC} = 0.7495$ and $\kappa_{BNC} = 0.7592$). However, in case of group based approach, we get poor agreement with the bug classes despite of the higher classification accuracy. For example, Naive Bayes classifier for Logic (L) group has a classification accuracy of 96.97%, but it has a $\kappa_{NBC} = 0.0672$ which indicates a poor agreement. However, after performing more analysis, we find that the number of bug samples is an important factor that influences the κ values and if we can test with more samples, κ value will be more meaningful.

ROC curve is a plot between *True Positive Rate (TPR)* and *False Negative Rate (FNR)* of a classification system and a system with higher *AUC* value is considered to be a better classification system. According to Fig 12, both Naive Bayes and Bayes net show almost similar values except two classes. However, Bayes net has an average *AUC* value which is higher than that of Naive Bayes classifier which indicates that Bayes net is relatively better in predicting true classes of the bug samples.

The next metric we consider for classifier performance is sample size versus classification accuracy curve. From 13, we can see that Bayes net's performance increases relatively faster compared to that of Naive Bayes classifier with the addition of new samples. We test with 28,266 bug samples; however, addition of more sample is likely to signify the performance differences between the classifiers.

We also consider the cost of misclassification and find that Bayes net is less prone of costly errors rather than Naive Bayes. Bayes net considers and tries to explore the conditional dependencies among the bug features that leads to a somehow semantic representation of dataset and prevents from major classification error. On the other hand, Naive Bayes classifier considers a fixed structure of class and attribute which may not reflect the data semantics and lead to major misclassification errors.

Thus, considering all of the above metrics for performance evaluation, we can conclude that, for *CVE* dataset, Bayes net classifier is preferable and it has more potential than Naive Bayes for classification with a little compromise in training cost (e.g., additional step for structure learning), but Naive Bayes is also a good choice for bug classification. We consider two approaches for bug classification in this research exploiting the domain knowledge about the bug class. From the above analysis and discussion, we can also conclude that if the domain knowledge is available about the bug class, the group-based approach with selected bug features is an appropriate solution, otherwise the general approach is always a reliable choice for classification.

B. Implication of Results

In our research, we apply two Bayesian classifiers for *CVE* dataset classification which is a novel idea. Our findings in the research have several implications. This section focuses on those implications.

1) Reduction of Effort: Although the feature extraction is still manual, we propose a machine learning based *automated* classification approach which can significantly reduce the human effort for classification task. The most interesting part of this approach is scalability; whereas manual bug classification is limited to the dataset size or availability of human experts, our approach can be scaled to any sized dataset with suitable feature extraction techniques.

2) Reduction of Cost: The proposed approach can significantly reduce the cost of misclassification in terms of time and money. Our approach provides a classification accuracy of 81.50% for *CVE* dataset which can be compared to the existing well-known result of 83% accuracy[13] for another dataset. Thus, if adopted, our approach can greatly help the triagers and the developers with accurate classification information.

3) Adaptive decision making: We propose an adaptive decision making approach for bug classification. That means, it is not restricted to only a fixed Naive Bayes classifier, rather it takes adaptive decision based on the availability of the domain knowledge about the bug class. If the triager can predict the group (i.e., level one class) of the bug sample, our algorithm can use the group-based approach for more targeted classification with higher accuracy. However, if the group information is not available (Fig 2-(i)), the proposed algorithm can adopt the general approach for classification considering all features. Thus, our proposed approach provides flexibility during classification and adapts with different available information.

4) New knowledge exploration: In section V-F, we show how Bayes net can be used to explore new knowledge about bug class and its features. The learned structure of Bayes net can be used to mine different hidden facts about the conditional dependencies which can be used to update the subjective interpretation of bug classes. For example, the current domain knowledge does not consider a conditional dependence relationship between *A8,specific offset of values* and *A28, Did not catch an exception* as they are from different bug groups; however, observing this fact would help the domain experts to rethink about the subjective interpretation of the bug classes.

VII. Future Works

In this research, we propose an automated bug classification approach for *CVE* dataset using Bayesian probabilistic approach. We successfully determined the applicability of the approach for the target dataset; however, we have some future plans with the existing works. They are discussed below.

- **Automated Bug Feature Extraction:** In this research, we use the extracted feature values by Billah and Roy [8] and they are extracted manually. We are planning to automate the feature extraction technique using some IR methods such as *TFIDF, Information gain (IG), Mutual Information (MI)* etc as well as sentiment analysis techniques.
- **Feature Values and Weights:** In this research, we consider binary values and uniform weights for all the features. In future, we will investigate whether the multinomial values and prioritized weights can better reflect the feature importance or not.
- **Reorganization of Bug features:** The learned Bayes nets in our experiments reveal certain important information about the conditional dependencies among the bug features. We will exploit that information and analyze the dataset more deeply to update or reorganize the bug features under the bug groups and classes.
- **Real World User study:** We also have a plan to conduct a real world user study to determine the effectiveness of our proposed approach for classification.

VIII. Summary

To summarize, we propose a machine learning based bug classification approach for *CVE* dataset with 28,266 bug samples, 22 classes and 64 bug features. We use two Bayesian classifiers such as Naive Bayes classifier and Bayes net classifier. In our approach, we conduct the experiments from two different view points -group based approach (i.e., exploits the subjective interpretation of bug class and considers selected features) and general approach (i.e., considers all 64 features). We analyze the experimental results and measure the performance of both classifiers using several performance metrics such as classification accuracy, κ statistic, *AUC* value, misclassification costs etc. From the analysis, we find that Bayes net classifier is preferable to Naive Bayes for classification with *CVE* dataset. We also find that group-based approach is applicable for bug classification when the bug group is known, otherwise general approach (i.e., considering all features) is a suitable choice for classification.

References

[1] Area Under an ROC Curve. URL http://gim.unmc.edu/dxtests/roc3.htm.

[2] Bayes net Learning. URL http://www.slideshare.net/gladysCJ/slidesPhDthesisGCastillo.

[3] Bayes net Search Algorithms. URL http://www.bayesnets.com/.

[4] Mitre Corporation: Common Vulnerabilities and Exposures. URL http://cve.mitre.org/data/downloads/index.html.

[5] Cohen Kappa Statistic. URL http://www.statstodo.com/CohenKappa_Exp.php.

[6] T. Aslam. A Taxonomy of Security Faults in the Unix Operating System. Master's thesis.

[7] K. Billah. Detecting Dissimilar Classes of Source Code Defects. Master's thesis, University of Saskatchewan, Canada, 2013.

[8] K. Billah and C.K. Roy. ExTax: A User Driven Classification Framework for Extensible Source Code Defect Taxonomies. Technical report, University of Saskatchewan, Department of Computer Science, 2012.

[9] R. Chillarege, I.S. Bhandari, J.K. Chaar, M.J. Halliday, D.S. Moebus, B.K. Ray, and M.-Y. Wong. Orthogonal Defect Classification-A Concept for In-Process Measurements. *TSE*, 18(11):943 –956, 1992.

[10] C. Chow and C. Liu. Approximating Discrete Probability Distributions with Dependence Trees. *TIT*, 14(3):462–467, 1968.

[11] D. E. Knuth. The Errors of TEX. *Journal of Software: Practice and Experience*, 7:607–685, 1989.

[12] T. Nakamura, L. Hochstein, and V.R. Basili. Identifying Domain-Specific Defect Classes Using Inspections and Change History. In *ISESE*, pages 346–355, 2006.

[13] Neelofar, M.Y. Javed, and H. Mohsin. An Automated Approach for Software Bug Classification. In *Proc. CISIS)*, pages 414 –419, 2012.

[14] C. B. Seaman, F. Shull, M. Regardie, D. Elbert, R.L. Feldmann, Y. Guo, and S. Godfrey. Defect Categorization: Making Use of a Decade of Widely Varying Historical Data. In *Proc. ESEM*, pages 149–157, 2008.

TABLE VI
BUG FEATURES

Group (Level 1)	Group Feature	Class Feature (Level 2)	
Computation (C)	Wrong output (A0)	Truncated values (A5) Rounded values (A7) Meaningless values (A9)	Expanded values (A6) Specific offset of values (A8) Undefined values (A10)
Logic (L)	Wrong Format (A1)	Missed a check (A11) Loop terminal condition wrong (A13) Loop terminal condition reversed (A15) Wrong Precedence (A17) Missed to save a value (A19) Missed to set the right value (A21) Extra complex logic (A23) Ran a condition as a tautology or contradiction (A25) Made invalid extra checks (A27) Caught the exception that will never be thrown (A29) Caught the right exception and did wrong handing (A31) Made some branch that will always be taken A(33) Made wrong interactions among code (A35) Changed Implementation from what was recommended (A37)	Made a wrong condition (A12) Loop terminal condition missing (A14) Wrong operator (A16) Wrong Operand A(18) Missed to Update a Value (A20) Missed the relation (A22)) Ran a loop more than it should (A24) Made more than required checks (A26) Did not catch an exception (A28) Caught the right exception, but didn't handle it (A30) Made some branch that will never be taken (A32) Made wrong connections with components (A34) Changed Algorithm (A36)
Memory (M)	Memory Error (A2)	Tried to access non-allocated memory (A38) Tried to access memory inside permitted range with wrong attitude (A40) Tried to deallocate memory that hasn't been allocated (A42) Did not deallocate a memory (A44) Deallocated only parts (A46) Did not supply proper values for interface (A48)	Tried to access memory out of permitted value (A39) Tried to free memory that was deallocated (A41) Tried deallocation in a wrong way (A43) Did not allocate a memory (A45) Allocated less than what's required (A47)
Data, Interface and I/O (D)	More than One Entities (A3)	Did not supply proper types of values for interface (A49) Tried to use an interface without enough data (A51) Tried to cast up or down (A53) Supplied extra Input (A55) Supplied wrong data format (A57) Failed to provide access to a member (A59)	Used default interface values (A50) Tried to assign a data piece to a non-matching one (A52) Changed data solely to meet the requirement at hand (A54) Did not supply enough input (A56) Provided access to members should not be accessed (A58)
Synchronization (S)	More than One Processes (A4)	Deadlock (A60) Out of Sync (A62)	Data Race (A61) Lock and Release (A63)