On the Use of Software Tracing and Boolean Combination of Ensemble Classifiers to Support Software Reliability and Security Tasks

Md. Shariful Islam

A thesis

in

the Department

of

Electrical And Computer Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of

Doctor Of Philosophy (Electrical And Computer Engineering) at

Concordia University

Montreal, Quebec, Canada

November 2020

CONCORDIA UNIVERSITY
School of Graduate Studies

This is to certify that the thesis prepared

By: **Md. Shariful Islam**

Entitled: **On the Use of Software Tracing and Boolean Combination of Ensemble Classifiers to Support Software Reliability and Security Tasks**

and submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy (Electrical and Computer Engineering)

complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

Dr. Chun Wang_____ Chair

Dr. Hanifa Boucheneb _____ External Examiner

Dr. Hassan Rivaz_____ Examiner

Dr. Nawwaf Kharma_____ Examiner

Dr. Roch H. Glitho_____ Examiner

Dr. Abdelwahab Hamou-Lhadj_____Thesis Supervisor

Approved by _____

Chair of Department or Graduate Program Director

_____ _____

Month/day/year Dean

Faculty of Engineering and Computer Science

Abstract

On the Use of Software Tracing and Boolean Combination of Ensemble Classifiers to Support Software Reliability and Security Tasks

Md. Shariful Islam, Ph.D. Candidate

Concordia University, 2020

In this thesis, we propose an approach that relies on Boolean combination of multiple one-class classification methods based on Hidden Markov Models (HMMs), which are pruned using weighted Kappa coefficient to select and combine accurate and diverse classifiers. Our approach, called WPIBC (Weighted Pruning Iterative Boolean Combination) works in three phases. The first phase selects a subset of the available base diverse soft classifiers by pruning all the redundant soft classifiers based on a weighted version of Cohen's kappa measure of agreement. The second phase selects a subset of diverse and accurate crisp classifiers from the base soft classifiers (selected in Phase1) based on the unweighted kappa measure. The selected complementary crisp classifiers are then combined in the final phase using Boolean combinations. We apply the proposed approach to two important problems in software security and reliability: The detection of system anomalies and the prediction of the reassignment of bug report fields.

Detecting system anomalies at run-time is a critical component of system reliability and security. Studies in this area focus mainly on the effectiveness of the proposed approaches -the ability to detect anomalies with high accuracy. Less attention was given to false alarm and efficiency. Although ensemble approaches for the detection of anomalies that use Boolean

combination of classifier decisions have been shown to be useful in reducing the false alarm rate over that of a single classifier, existing methods rely on an exponential number of combinations making them impractical even for a small number of classifiers. Our approach is not only able to maintain and even improve the accuracy of existing Boolean combination techniques, but also significantly reduce the combination time and the number of classifiers selected for combination.

The second application domain of our approach is the prediction of the reassignment of bug report fields. Bug reports contain a wealth of information that is used by triaging and development teams to understand the causes of bugs in order to provide fixes. The problem is that, for various reasons, it is common to have bug reports with missing or incorrect information, hindering the bug resolution process. To address this problem. researchers have turned to machine learning techniques. The common practice is to build models that leverage historical bug reports to automatically predict when a given bug report field should be reassigned. Existing approaches have mainly relied upon classifiers that make use of natural language in the title and description of the bug reports. They fail to take advantage of the richly detailed sequential information that is present in stack traces included in bug reports. To address this, we propose an approach called EnHMM which uses WPIBC and stack traces to predict the reassignment of bug report fields.

Another contribution of this thesis is an approach to improve the efficiency of WPIBC by leveraging the Hadoop framework and the MapReduce programming model. We also show how WPIBC can be extended to support heterogenous classifiers.

Acknowledgement

I would like to express my special thanks of gratitude to Dr. Wahab Hamou-Lhadj who gave me the golden opportunity to do this wonderful research by accepting to enroll me as a Ph.D. student. He pushed me when I needed to be pushed and complimented me on jobs well done. I thank him for sharing his vast knowledge and expertise in this area, his trust in me, and for keeping his door always open for helpful feedback and conversation. More than anyone else, his influence has contributed to my development as a scientist.

I would like to express my thank to Dr. Wael Khreich and Dr. Abdelaziz Trabelsi for imparting their knowledge and expertise in this study. I am also grateful to thank my committee members for taking the time to read this dissertation and to serve on my thesis committee.

I would like to thank everyone at the Research Lab of Professor Hamou-Lhadj for their friendship, encouragement, and stimulating discussions.

I would also like to thank NSERC (Natural Science and Engineering Research Council of Canada), the Gina Cody School of Engineering and Computer Science, and the Faculty of Graduate Studies for their generous financial support.

I want to thank my lovely parents, wife, brothers and friends who helped me a lot in ups and downs, successes and failures with their encouragements and warm support. None of these could be possible without them.

Table of Contents

List of Figures

List of Tables

List of Acronyms

- *5FCV:* 5-Fold Cross Validation
- *ADFA-LD:* ADFA Linux Dataset
- *ADS:* Anomaly Detection System
- *AUC:* Area Under the Curve
- *BBC2:* Pair-wise Brute-force Boolean Combination
- *BICKER:* Kappa-pruned Ensemble-based Iterative Boolean Combination Rules
- *BR:* Bug Reports
- *BW:* Baum-Welc
- *CANALI-WD:* CANALI Windows Dataset
- *EM:* Expectation-Maximization
- *EnHMM:* Ensemble of Hidden Markov Models
- *FB:* Forward-Backward
- *HDFS:* Hadoop Distributed File System
- *HIDS:* Host-based Intrusion Detection System
- *HMM:* Hidden Markov Model
- *IBC:* Iterative Boolean Combination
- *Im-ML.KNN:* multi-label imbalanced KNN
- *KNN:* K Nearest Neighbor
- *MASKED:* A MapReduce Solution for the Kappa-pruned Ensemble-based Anomaly Detection System
- *ML*.KNN: multi-label KNN
- *NIDS:* Network Intrusion Detection System
- *OCSVM:* One-Class Support Vector Machine
- *PBC:* Pruning Boolean Combination
- *ROC:* Receiver Operating Characteristics
- *ROCCH:* Receiver Operating Characteristics Convex Hull
- *STIDE:* Sequence Time-Delay Embedding
- *WPIBC:* Weighted Pruning Iterative Boolean Combination
- *avg:* average value on 5-Fold Cross Validation results
- *fpr:* false positive rate outputted by a crisp classifier
- *kp-fpr:* kappa versus false positive rate plotting diagram
- *kp-tpr:* kappa versus true positive rate plotting diagram
- *max:* maximum value on 5-Fold Cross Validation results
- *min:* minimum value on 5-Fold Cross Validation results
- *std:* standard deviation on 5-Fold Cross Validation
- *tpr:* true positive rate outputted by a crisp classifier

Chapter 1. Introduction

1.1 Anomaly Detection Systems

Intrusion Detection Systems (IDS) are divided into two categories: Network Intrusion Detection Systems (NIDS) and Host-based Intrusion Detection Systems (HIDS). A NIDS monitors and analyzes network traffic. It is transparent (i.e., it can move in different locations) and independent (i.e., it can work in different network topologies). An HIDS works on a host computer and monitors user activities to detect unauthorized access, illegitimate modification of configuration files, and other unwanted behaviors. IDS can be further classified into two categories: Signature-based (or misuse) IDS and Anomaly Detection Systems (ADS). The former can only detect known attacks [1], whereas the latter, the focus of this thesis, is capable of detecting novel attacks by analyzing deviations from the normal behavior of a system.

An ADS is trained offline in a safe environment using data collected from running the system in a normal threat-free environment. The resulting model is put in operation. When the ADS notices an abnormal activity that deviates from the trained model, it raises an alert of a possible attack on the system.

Anomaly detection refers to the problem of finding unexpected patterns of system or user generated data that do not conform to a preestablished normal behavior [2]. The last two decades have seen an increase in attention to the field of anomaly detection with the emergence of several approaches using a panoply of methods including statistical methods, machine learning, and data mining (e.g., [3] [4] [5] [6]). Although these techniques vary in their design, the common practice is to build a model that represents the normal behavior of a system, which can later be used to

detect deviations from normalcy.

In software security, system anomalies may be due to attacks or misuse of resources. To detect these attacks, most anomaly detection techniques use the temporal order of system calls generated by processes at the kernel level as features to train an anomaly detection model [3] [7] [8] [9]. In recent years, ensemble approaches that combine the decisions of multiple crisp classifiers[1] using Boolean combination rules have been shown to improve significantly the prediction accuracy, while reducing the false alarms rate, hence increasing the general adoption of anomaly detection techniques in practice [10] [11]. However, an exhaustive brute-force search to determine optimal combinations leads to an exponential number of combinations, which is prohibitive even for a small number of classifiers [10]. To address this issue, Khreich et al. [11] proposed an Iterative Boolean Combination (IBC) approach for combining relatively a large number of soft Hidden Markov Model (HMM) classifiers while avoiding the exponential explosion of a Pair-wise Brute-force Boolean Combination (BBC2) [10]. The problem is that IBC produces a sequence of combination rules that grows linearly with both the number of soft HMM classifiers and the number of iterations, hindering it difficult to analyze and understand. Furthermore, the algorithm is sensitive to the order of the combined crisp HMM classifiers, making it challenging to find the best subset for combination operations.

To reduce the computation time and complexity of BBC2, Soudi et al. [12] proposed a Pruning Boolean Combination (PBC) approach. In short, PBC prunes all trivial (a crisp classifier that produces always either positive or negative) and redundant crisp classifiers and then selects

[1]A crisp classifier is the one that gives a decision (e.g., positive or negative) instead of scores (i.e., likelihood probability or similarity). This is contrasted with a soft classifier, which produces scores instead of a decision. A soft classifier can be converted into one or more crisp anomaly classifiers by setting different thresholds on the output scores [6][7].

complementary crisp classifiers based on the Kappa agreements between each crisp classifier's decisions and the true labels (ground truth) on the validation set. PBC improves the efficiency of BBC2, however, it cannot ensure the diversity among soft HMM classifiers. For example, if the scores of a subset of available soft HMM classifiers on a validation set are almost the same, the responses of the crisp HMM classifiers at a decision threshold of these redundant soft HMM classifiers will probably be the same. Particularly, the computed kappa values for each crisp HMM classifiers generated from these redundant soft HMM classifiers will probably be almost equal. So, if the kappa value of one of these redundant crisps HMM classifiers is close to Min or Max, the chances of selecting the remaining redundant crisp HMM classifiers are very high. Therefore, only one soft HMM classifier from this subset of redundant soft HMM classifiers should be used while the rest of the redundant soft HMM classifiers should be pruned before converting them into crisp HMM classifiers.

In addition, although PBC reduces the computation time and complexity of BBC2, it cannot ensure the diversity among the combined soft classifiers, despite the fact that the performance of an ensemble method has been shown to be highly dependent on the diversity of the combined classifiers [13] [14].

Thesis Statement:

In this thesis, we propose a weighted Kappa-pruned ensemble approach, called Weighted Pruning Iterative Boolean Combination (WPIBC). WPIBC selects the most diverse classifiers from a set of candidate classifiers while pruning the redundant ones. Then, we leverage Boolean combination techniques ([10] [11]) to combine the decisions produced by each selected diverse classifier. We compare our approach with the existing BBC2 [10], IBC [11], and PBC [12] Boolean

combination techniques. The results show that WPIBC outperforms BBC2, IBC, and PBC by achieving better accuracy with lower false positive rate and also significantly reducing the computation time as well. First, we evaluate WPIBC by applying it to the detection of system anomalies using datasets of traces of system calls. Further, we evaluate WPIBC by applying it to the prediction of the reassignments of bug report fields using datasets of traces of function calls datasets. Another contribution of this thesis is an approach that improves the efficiency of WPIBC using the MapReduce programming model.

1.2 Thesis Contributions

We organize this thesis in three contributions that are depicted in Figure 1.1.

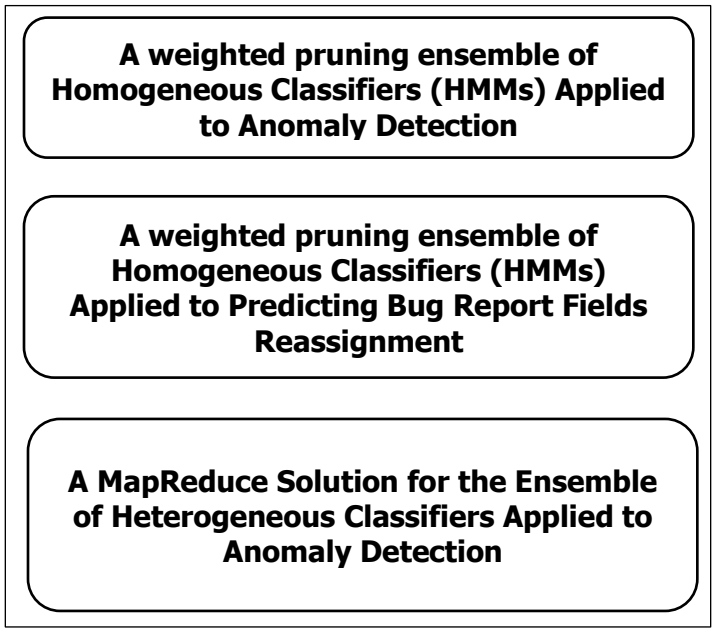


Figure 1.1. Research Contributions

Contribution 1: A weighted pruning ensemble of HMMs for detecting system anomalies

In this work, we propose weighted pruning-based Boolean combination approach for selecting and combining accurate and diverse anomaly classifiers. Our approach works in three phases. The

first phase selects a subset of the available base diverse soft classifiers by pruning all the redundant soft classifiers based on a weighted version of Cohen's kappa measure of agreement. The second phase selects a subset of diverse and accurate crisp classifiers from the base soft classifiers (selected in Phase1) based on the unweighted kappa measure. The selected complementary crisp classifiers are then combined in the final phase using Boolean combinations. The results on two large scale datasets show that the proposed weighted pruning approach is able to maintain and even improve the accuracy of existing Boolean combination techniques, while significantly reducing the combination time and the number of classifiers selected for combination.

Contribution 2: A weighted pruning ensemble of HMMs for predicting the reassignment of bug report fields

In this work, we leverage our ensemble HMMs for predicting the reassignment of Bug Report (BR) fields, another important problem in software reliability. Our approach, called EnHMM, is based on WPIBC by leveraging the natural ability of HMMs to represent sequential data to model the temporal order of function calls in BR stack traces. When applied to Eclipse and Gnome BR repositories, EnHMM achieves an average precision, recall, and F-measure of 54%, 76%, and 60% on Eclipse dataset and 41%, 69%, and 51% on Gnome dataset. We also found that EnHMM improves over the best single HMM by 36% for Eclipse and 76% for Gnome. Finally, a comparative study shows that EnHMM outperforms leading BR field reassignment prediction methods.

Contribution 3: A MapReduce solution for the ensemble of heterogeneous classifiers

In this contribution, we leverage heterogeneous machine learning techniques and big data platform for improving both the accuracy and the efficiency of the propose weighted pruning

ensemble approach. The propose MapReduce Solution for the Kappa-pruned Ensemble based Anomaly Detection System (MASKED) profiles the heterogeneous features from large-scale traces of system calls and processes them by heterogeneous anomaly classifiers which are Sequence-Time Delay Embedding (STIDE), Hidden Markov Models (HMMs), and One-class Support Vector Machine (OCSVM). We deployed MASKED on a Hadoop cluster using the MapReduce programming model. We compared their efficiency and scalability by varying the size of the cluster. We assessed the performance of the proposed approach using the CANALI-WD dataset which consists of 180 GB of execution traces, collected from 10 different machines. Experimental results show that MASKED becomes more efficient and scalable as the file size is increased (e.g., 6-node cluster is 8 times faster than the 2-node cluster). Moreover, the throughput achieved on a 6-node solution is up to 5 times better than a 2-node solution.

1.3 Thesis Organization

The rest of the thesis is organized as follows. Chapter 2 surveys the state of the art in ensemble-based anomaly detection approaches and predicting the reassignment of bug report fields. Chapter 3, 4, and 5 explains the three contributions in this thesis we reported in Section 1.3. The closing Chapter 6 highlights the thesis and gives the future directions and concluding remarks.

1.4 Related Publications

1. M. S. Islam, W. Khreich and A. Hamou-Lhadj, "Anomaly Detection Techniques Based on Kappa-Pruned Ensembles," IEEE Transactions on Reliability, vol. 67, no. 1, pp. 212-229, 2018. [15]

2. M. S. Islam, K. K. Sabor, A. Hamou-Lhadj, A. Trabelsi and L. Alawneh, "MASKED: A

MapReduce Solution for the Kappa-pruned Ensemble-based Anomaly Detection System," in the 18th IEEE Int. Conf. on Software Quality, Reliability, and Security, Lisbon, Portugal, 2018. [16]

3. M. S. Islam, A. Hamou-Lhadj, K. K. Sabor, M. Hamdaqa and H. Cai, EnHMM: On the Use of Ensemble HMMs and Stack Traces to Predict the Reassignment of Bug Report Fields, (in preparation). [17]

Other Publications:

4. N. Ebrahimi, A. Trabelsi, M. S. Islam, A. Hamou-Lhadj and K. Khanmohammadi, "An HMM-based approach for automatic detection and classification of duplicate bug reports," Information and Software Technology, Elsevier, vol. 113, pp. 98-109, 2019. [18]

5. N. Ebrahimi, M. S. Islam, A. Hamou-Lhadj and M. Hamdaqa, "An Effective Method for Detecting Duplicate Crash Reports Using Crash Traces and Hidden Markov Models," in Proc. of the IBM 26th Annual International Conference on Computer Science and Software Engineering (CASCON'16), 2016. [19]

Chapter 2. Background and Literature Review

2.1 ROC-Based Boolean Combination of Multiple Classifiers

Ensemble methods have been proposed to improve the overall accuracy by combining the outputs of several accurate and diverse models [8] [20] [21] [22]. In particular, combining the outputs from multiple crisp HMM (Hidden Markov Model) classifiers generated from multiple soft HMM classifiers, each trained with a different number of states, in the ROC space, has been shown to provide a significant improvement in the detection accuracy of system call anomalies [11] [23] [24]. The following sub-sections addressed existing Boolean combination techniques on the ROC space such as BBC2 [10], IBC [11] and one recent pruning based PBC [12] with their limitations.

2.1.1 The ROC Convex Hull (ROCCH)

All the points in a ROC space can be classified into two groups superior and inferior based on their *tpr* and *fpr*. Suppose a and b are two operating points in the ROC space, a is defined as superior to b, if $fpr_a \leq fpr_b$ and $tpr_a \leq tpr_b$. If a ROC curve has $tpr(*) > fpr(*)$ for all its points $(*)$, then it is a proper ROC curve. The ROC convex hull (ROCCH) is therefore the piece-wise outer envelope connecting only its superior points [10] [25]. The linear interpolation is used to connect the two adjacent superior points so that, no points in a ROC space lies out of the final ROCCH curve. The accuracy of a ROCCH curve is measured by the Area Under the Curve (AUC). The ROCCH can be used for the combination of two or more crisp classifiers in a ROC space [10] [11]. However, ROCCH combination rules discard the inferior points without verifying their combination in order to improve the system performance.

2.1.2 The Boolean Combination of ROC Curves

The very first Boolean combination approach, proposed by Daugman [26], used only the conjunction (AND) and disjunction (OR) rules and fused on all the responses in a ROC space. The author applied these rules in a biometric test and concluded that the new composite ROCCH may increase the AUC of the ROC curve. As a consequence, other researchers also applied the AND or OR combination to combine soft classifiers [27] [28]. For example, consider a pair of soft classifiers $(S_a,\ S_b)$ and the various decision thresholds are T_a and T_b, respectively. In a pair-wise combination, the AND or OR rules are fused between each pair of converted crisp classifiers $\left(C_i^a, C_j^b\right)$. The optimum thresholds are then selected based on the Neyman-Person test[2] [29]. Finally, the selected optimum thresholds along with the corresponding Boolean functions are stored and used during operation.

However, the AND and OR combinations cannot provide optimal thresholds when the training and validation datasets are limited and imbalanced [11]. Because, the resulting ROC curves using the limited and imbalance data may lead to the appearance of large concavities [30]. In particular, the false alarm may be increased, if we fuse the best classifier and the worst classifier. But, the diversity among the combined classifiers is an important factor in order to improve the performance while reducing the false alarm rate [14]. Therefore, further improvement is possible by including the other Boolean rules, in addition to the AND and OR rules. The following sub-sections introduce the three most common combination techniques using all Boolean rules: Pair-wise Brute-force Boolean Combination (BBC2) [10], Iterative Boolean Combination (IBC) [11] and Pruning Boolean Combination (PBC) [12]. We also report on the limitations and complexities

[2] The point (tpr_{opt}, fpr_{opt}) of a crisp classifier in a ROC space, is optimum, if all the other points for the same value of fpr_{opt}, the value of tpr_{opt} is maximum.

of these techniques.

2.1.3 The Pair-wise Brute-force Boolean Combination (BBC2)

The Pair-wise Brute-force Boolean Combination (BBC2) fuses all possible pairs of crisp classifiers generated from all the available soft classifiers using all Boolean functions. For example, Figure 2.1 shows two soft HMM classifiers, D1 and D2, which have four corresponding crisp classifiers (i.e., single points on the ROC curve), obtained by setting four different thresholds on scores computed by D1 and D2. The two soft classifiers, D1 and D2, produced two ROC curves where each has four candidate crisp classifiers: D1(c1, c2, c3, and c4) and D2(p1, p2, p3, and p4). The Area Under the Curve (AUC) of the ROC curve produced by the soft classifier D1 is 0.82, and D2 is 0.62, meaning that D1 performs better than D2.

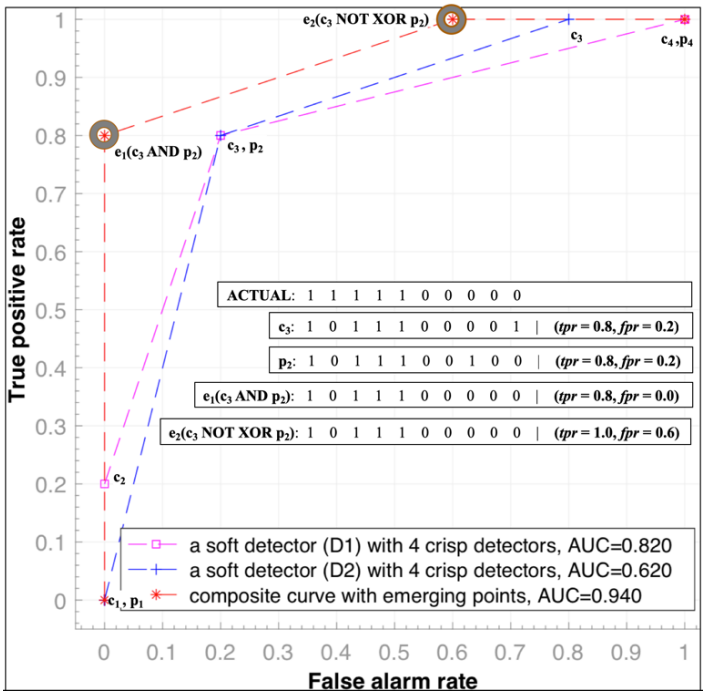


Figure 2.1. An example of Boolean combination of HMMs

Since BBC2 uses all possible combination pairs among all the available candidate crisp classifiers, the eight candidate crisp classifiers (in Figure 2.1, c1 to c4 and p1 to p4) produce 66 combination pairs. Each pair is then combined by ten different Boolean functions (a∧b, ¬a∧b, a∧¬b, ¬(a∧b), a∨b, ¬a∨b, a∨¬b, ¬(a∨b), a⊕b, a≡b). Therefore, it produces 66x10=660 emerging responses on the ROC space, which are then turned into 660 emerging points (e) on the ROC space. The points that have the highest AUC are then selected to compute the target composite ROC curve. In this example, two emerging points, e1 and e2 are used to compute the final composite ROC curve that improves the AUC.

As BBC2 uses all Boolean functions, it implicitly combines responses of both accurate and diverse crisp classifiers at both superior and inferior points in the ROC space. However, the pair-wise brute-force strategy is computationally expensive due to the high number of permutations. For example, if the number of crisp classifiers is N, there are N^2 possible combinations for only one Boolean function. Barreno et al. [10] reported that exploiting all Boolean functions using an exhaustive brute-force search to determine optimum points leads to an exponential number of combinations.

2.1.4 The Iterative Boolean Combination (IBC)

IBC avoids the impractical exponential explosion associated with the BBC2 by combining the emerging responses on a composite ROCCH sequentially. It first combines the first two ROC curves of the first two soft classifiers. Then, the combined ROCCH, particularly, the emerging points are combined with the next ROC curve, and so on until the K^{th} ROC curve is combined. IBC repeats these sequential combinations iteratively until there are no further improvements or it reaches to a predefined maximum number of iterations. However, in practice, IBC requires a

sequence of combinations of 11 to 20 crisp classifiers to reach a final point on the final composite ROCCH [12]. In fact, it grows linearly with the increase of the number of iterations. Because of this sequence of combination rules, IBC is more complex to analyze and understand during testing time. Moreover, the order of combined crisp classifiers makes the IBC algorithm more sensitive to finding the best subset.

It is evident that the computation time and complexity increase exponentially for BBC2 and linearly for IBC with the increase of the number of combined soft classifiers (K), and thus making them inefficient.

2.1.5 The Pruning Boolean Combination (PBC)

To reduce the computational complexity, Soudi et al. [12] proposed a Pruning Boolean Combination (PBC) approach. In short, PBC prunes all trivial (a crisp classifier that produces always either negative or positive) and redundant crisp classifiers and, then, selects complementary crisp classifiers based on the kappa agreements between each crisp classifier's decisions and the true labels (ground truth) on the validation set. The MinMax-Kappa (a pruning technique of PBC) computes the kappa values for all possible crisp HMM classifiers, and then sets Min (Minimum kappa value) and Max (Maximum kappa value) boundaries with sorting them in ascending order. After that, MinMax-Kappa selects *m* complementary crisp classifiers where 50% or m/2 crisp HMM classifiers whose kappa values are close to Min and another 50% or m/2 crisp HMM classifiers whose kappa values are close to Max.

However, PBC uses the kappa coefficients between two crisp HMM classifiers, it cannot ensure the diversity among soft HMM classifiers. For example, if the scores of a subset of available soft HMM classifiers on a validation set are almost the same, the responses of the crisp HMM

classifiers at a decision threshold of these redundant soft HMM classifiers will probably be the same. Particularly, the computed kappa values for each crisp HMM classifiers generated from these redundant softs HMM classifiers will probably be almost equal. So, if the kappa value of one of these redundant crisp HMM classifiers is close to Min or Max, the chances of selecting the rest of the redundant crisp HMM classifiers are very high. Therefore, only one soft HMM classifier from this subset of redundant soft HMM classifiers should be used while the rest of the redundant soft HMM classifiers should be pruned before converting them into crisp HMM classifiers.

To ensure the diversities among the combined crisp classifiers, we proposed a more sophisticated pruning technique that selects the smallest and most diverse subset of classifiers (among all available ones), which does not only reduce the computation time and complexity for Boolean combinations but also maintains or improves the detection accuracy (while reducing the false alarm rate) using the smallest number of Boolean combinations. We validated the proposed pruning-based ensemble approach by applying on two diverse applications: anomaly detection systems and predicting the bug report fields reassignments.

2.2 Review on Anomaly Detections Systems (ADS)

2.2.1 Introduction

Anomaly detection refers to the problem of finding unexpected patterns of system or user generated data that do not conform to the normal behavior. In data mining, anomalies are the things that do not conform to any normal events or items or observations in a normal dataset. Generally, we can say any unexpected patterns of data that do not conform any right of normal behavior referred to as anomalies, outliers, attacks, novelties, noises, deviations and exceptions. Chandola et al., [2] defines an anomaly is a pattern that does not fit to a well-defined manner of normal

behavior. However, a well-trained anomaly classifier may also have an incomplete view of the original normal process behaviors due to the limitation of training samples. This kind of incomplete view of the actual normal process behavior leads to misclassifying rare normal events, and thus, raises the false alarms. For example, in Figure 2.2, the points N_1 and N_3 are correctly classified but a rare normal event, N_2 is misclassified.

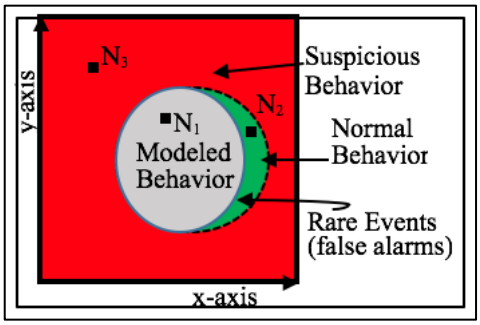


Figure 2.2. A simple example of anomalies

Therefore, modeling a precise normal behavior using a limited normal dataset is very difficult due to the following key challenges:

- Defining a normal region for every possible normal behavior is very difficult.
- The border between normal and anomalous behavior is often not precise.
- Normal behavior can be changed over time.
- The degree of an anomaly is application specific (e.g., in medical, a very small deviation reports an anomaly, but this might be considered normal in stock)
- Difficult to overcome ambiguous anomaly due to the uncertain noises in data
- The availability of labeled data for training/validation of models

Anomalies can be classified into three types: point anomalies, contextual anomalies, and collective anomalies [2].

Point anomalies-in which an individual instance can be considered as an anomaly with respect to one or more normal behaviors. For example, in Figure 2.2, points N_2 and N_3 are an anomaly with respect to normal regions. In that case, vector data instances can be handy for detecting point anomalies [31].

Contextual anomalies-in which an individual instance can be considered as an anomaly with respect to a specific context, not otherwise [32]. For example, in a testing sequence *A B A B* where at third and fourth positions, A calls to B, which does not appear in normal behavior for that specific positions, although it is appear for other positions in the rest of the normal sequences. However, in most cases, using a single specific context raises the false alarms. In fact, using two or more specific contexts and integrating their outputs also increase the computation time and complexities.

Collective anomalies-in which a collection of targeted data instances is anomalous with respect to the entire data set. When the individual data instances may be considered as normal, but their appearance together as a collection is anomalous [33]. For example, in a human electrocardiogram output, the presence of a same low value for a while is reported as an anomaly with respect to a long time-series input data instance.

Based on the availability of data labels (normal/anomaly), the whole ADS techniques can be classified into three different models [2]:

Supervised ADS: Techniques that train the models using both available labeled normal and

anomalous classes of data instances are called supervised ADS. Although supervised ADS show lower false alarms, the major challenge is the use of imbalanced training dataset [34]. Particularly, the ratio of anomalous instances is far lower than the normal instances in the training data. Another major drawback is that supervised ADS models cannot detect novel attacks.

Semi-Supervised ADS: Techniques that train the models using only the available labeled normal class of data instances are called semi-supervised ADS. Since they do not use the anomalous class, they are more widely acceptable than the supervised ADS techniques [2]. In fact, they can detect even novel attacks. We also reviewed so far, some best semi-supervised ADS techniques, in fact, the contributions of this proposal are also an integration these techniques only.

Unsupervised ADS: Techniques that define the models with the assumption that the class labels for all the available data instances are unknown. Particularly, instead of using a training dataset, they use all data instances and implicitly assume that the normal class are far more common than anomalous class. When this assumption is true such models are the best choice, otherwise they may account a high false alarm rate.

Detecting anomaly is a binary classification problem. Based on the outputs of anomaly classifiers, the classifiers can be further classified into soft and crisp classifiers. Classifiers that produce scores instead of a decision (i.e., normal or anomaly) for a new test instance are called soft anomaly classifiers. On the other hand, classifiers that produce decisions (i.e., normal or anomaly) are called crisp anomaly classifiers. We can convert a soft anomaly classifier to one or more crisp anomaly classifiers by setting one or more thresholds (θ) on the output scores produced by a soft classifier. A crisp classifier always gives a decision whether the testing sample is normal ($score \geq \theta$) or anomalous ($score < \theta$) based on a predefined threshold, θ.

The ROC curve is a commonly used metric for evaluation of classifiers' performance. It plots the performances of a binary classifier in a 2-D space [25], where, y-axis represents the true positive rate (*tpr*) and x-axis represents the false positive rate (*fpr*) for every possible crisp classifier. The *tpr* is the proportion of correctly classified positive responses over the total number of positive samples tested by a crisp classifier. The *fpr* is the proportion of incorrectly classified negative responses over the total number of negative samples tested by a crisp classifier. Therefore, a single crisp classifier plots a single point (*fpr*, *tpr*) in a ROC space, while a soft classifier produces a ROC curve by connecting all the possible crisp classifier's points at various decision thresholds.

2.2.2 Background

Anomaly detection is used in a wide variety of applications such as fraud detection for credit cards, insurance or health care, intrusion detection for cyber-security, etc. [2]. The last two decades have seen an increase in attention to the field of anomaly detection. Several approaches have emerged using panoply of methods including statistical methods, machine learning, and data mining (e.g., [3] [4] [5] [6]). Although these techniques vary in their design, the common practice is to build a model that represents the normal behavior of a system, which can later be used to detect deviations from normalcy. Most anomaly detection techniques use the temporal order of system calls, generated by a process at the kernel level, as features [3] [7] [8] [9]. In security, system anomalies may be due to attacks. Detecting them is therefore an important task that can enhance system reliability. Shariyar et al. [35] presented an approach and a supporting tool to detect program functions that are likely to introduce faults in a software system by examining historical execution traces. Their approach can be used to enhance testing and other software verification methods. In a recent study, Sha et al. [36] proposed an approach based on anomaly detection to ensure the safety of cloud-based IT infrastructures. Bovenzi et al. [37] proposed an

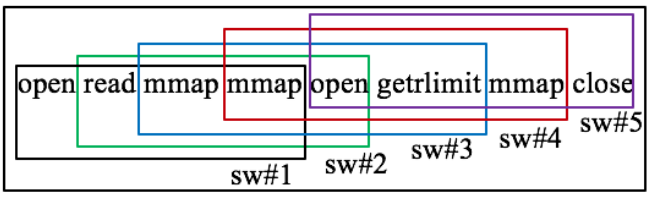
approach for revealing anomalies at the operating system level to support online diagnosis activities of complex software systems. Yang et al. [38] proposed an efficient method for detecting abnormal executions of Java programs using sequential pattern mining. Gizopoulos et al. [39] argued that large investment in the design and production of multicore processors may be put at risk because of reliability threats, mainly due to the existence of bugs and vulnerabilities, unless these systems are equipped with robust anomaly detection tools. They proposed multicore processor architectures that integrate solutions for online error detection, diagnosis, recovery, and repair during field operation.

2.2.3 Simple Sequence Matching Techniques using System Call Sequences

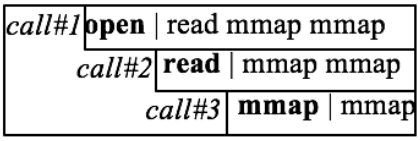
To our knowledge, the very first approaches for anomaly detection are based on sequence matching [36] [40] [41] [42]. During training, these approach builds the normal profile by segmenting the full-length sequences of system calls into a fixed-length contiguous sub-sequences using a fixed-size sliding window, shifted one by one symbol. An example with window size four is shown in Figure 2.3 (a). In testing, an unknown sequence of system calls is also segmented into sub-sequences (as in training) and classified as normal if all sub-sequences are present in the normal profile. Otherwise, it is classified as an attack.

We introduced two early simple techniques: *time-delay embedding (tide)* [33] and *Sequence Time Delay Embedding (STIDE)* [41]. The former one uses *lookahead* pairs to construct the normal database, whereas the later one uses continuous sub-sequences with a fixed window size. An example of *tide* and *STIDE* is illustrated in Figure 2.3. Let say, there are five distinct system calls: *open, read, mmap, getrlimit, and close;* and a sample sequence with length of eight. If the size of sliding window is $k=4$, we get five continuous sub-sequences, given in Figure 2.3 (a). The

lookahead pairs expends each sliding window (sw#) and records each call that follows at positions 1, 2, up to k-1. An expended lookahead table for sw#1 is shown in Figure 2.3 (b), where three call# are made by three *open*, *read*, and *mmap* system calls. For each call# (i.e., system call), the following system call(s) with their respected position(s) are then recorded into a normal database. Figure 2.3 (c) shows the final normal database constructed using five expended sliding windows (sw#) respectively. Similarly, instead of using *lookahead* pair, STIDE uses all unique sliding windows (sw#) or unique sub-sequences with a fixed window size to construct the normal database. Another key difference of STIDE is the storing technique. STIDE stores all unique sub-sequences using tree data structure. Figure 2.3 (d) shows the constructed normal database where each system calls acts as a root for each tree.



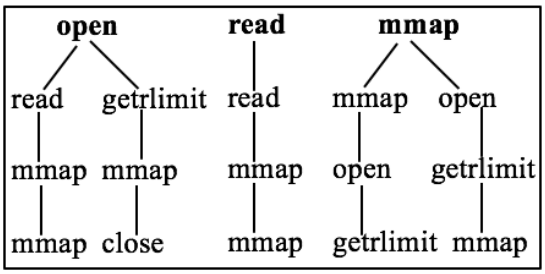
(a) sliding windows (sw#)



(b) expended each calls within sw#1

call	position1	position2	position3
open	read	mmap	mmap
	getrlimit		close
read	mmap	mmap	open
mmap	mmap	open	getrlimit
	open	getrlimit	mmap
	close		
getrlimit	mmap	close	
close			

(c) constructed normal database of *tide*



(d) constructed normal database of *STIDE*

Figure 2.3. An example of construction of normal database for *tide* and *STIDE*

Once the normal database is constructed, the next step is to detect the class label (normal/anomaly) of an unknown sequence. In matching measure, *tide* simply counts the number

of mismatches for all the sub-sequences generated from an unknown sequence. However, any mismatches are important as *tide* assumes the normal database covers most variations. But it is not wise to construct a normal database with all variations. Because rare anomalous could be detected as normal and thus false alarm will be increased. The solution is to use a threshold that acts as a boundary for a normal behavior in a system. *STIDE* uses *Hamming distance* as a metric of matching measure between two sequences and computes a score instead of a decision. Let say, an unknown sequence with *m* sub-sequences and a normal database with *n* sub-sequences. First they compute the minimum distances min_i for each sub-sequences *i{i=1,…m}* to a set of *n* normal sub-sequences *{j=1,…n}*. Then, they compute the maximum of the minimum distances as a score (*s*) for an unknown sequence using equation (2.1). Finally, they normalized the score $\hat{s} = s/L$, in order to make it independent over the sequence length *L*.

$$s = \underset{i=1}{\overset{m}{max}} \left\{ \underset{j=1}{\overset{n}{min_i}} \{d(i,j)\} \right\} \qquad (2.1)$$

In comparison, *STIDE* requires less in-memory and thus faster, because sub-sequences are stored as tree. In fact, *STIDE* accounts more discrimination and compact. However, using a single threshold on scores *s* generates an excessive number of false alarms that limits its deployment in commercial settings [33]. Moreover, typically, one complete trace generates a long sequence of system calls that increases the computation time due to a large number of sub-sequences. We also use STIDE as one of the heterogeneous soft classifiers and optimize these issues. We set different thresholds on the scores of STIDE to transform it into all possible crisp classifiers. Then these crisp classifiers are fed as an input.

2.2.4 Hidden Markov Models (HMMs) using System Call Sequences

HMM has been shown to be a very effective method to model a system's behavior over time [43]. Particularly, in detecting system anomalies using traces of system calls, HMMs outperforms the other approaches [24]. We also use HMMs as the base models in our proposed ensemble approach. An HMM is a stochastic model for sequential data determined by the two interrelated mechanisms–a latent Markov chain having a finite number of states and a set of observation probability distributions, each one associated with a state. An HMM is typically determined by three parameters $\lambda = (A, B, \pi)$, which represent the states and transition probability distribution (A) of a system in a Markov process, the observation probability distribution (B) of observation sequences that come from the temporal order of executions of a system, and the initial state probability distribution (π) of each hidden state in a Markov process. The first parameter, A, is usually hidden in an HMM. The only physical events are the observation sequence (B) that is associated with the hidden states of a Markov process. Figure 2.4 illustrates a generic topology of an HMM, $\lambda = (A, B, \pi)$ [40].

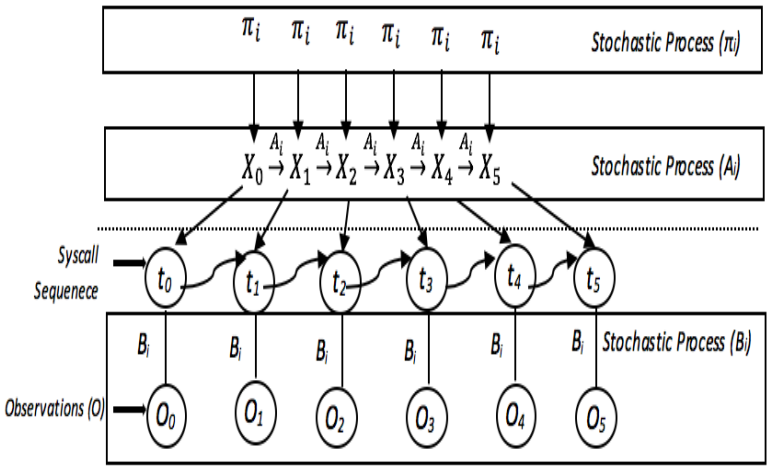


Figure 2.4. A general topology for an HMM model

Number of Hidden States (N): To learn an HMM, we have to set the number of hidden states

(N) in a Markov process. Let the distinct states be $S_i,\ i = \{0,1, \dots, N-1\}$. The notation $X_t = S_i$ represents the hidden state sequence at time t.

Number of Observation Symbols (M): To learn an HMM, we have to set the number of observation symbols (*M*). Let the distinct observation symbols be $R_k,\ k = \{0,1, \dots, M-1\}$. The notation $O_t = R_k$ represents the observed symbol R_k at time *t* for the given observation sequence $\mathcal{O} - (\mathcal{O}_0, \mathcal{O}_1, \dots, \mathcal{O}_{T-1})$, where T is the length of the observation sequence.

State Transition Distribution (A): The first-row stochastic process is the hidden state transition probability distribution matrix $A = \{a_{ij}\}$. A is an $N \times N$ square matrix and the probability of each element $\{a_{ij}\}$ is denoted in equation (2.8) as:

$$a_{ij} = P\big(state\ S_j\ at\ t+1 \big| state\ S_i\ at\ t\big), \qquad (2.8)$$

$$i, j = \{0,1, \dots, N-1\}$$

The transition from one state to the next is a Markov process of order one [44]. This means the next state depends only on the current state and its probability value. As the original states are "hidden" in HMM, we cannot directly compute the probability values in the past. But we are able to observe the observation symbols for the current state S_i at time t from a given observation sequence $\mathcal{O}$ to learn an HMM model.

Observation Symbol Distribution (B): The second-row stochastic process is the observation symbol probability distribution matrix $B = \{b_j(R_k)\}$. B is an $N \times M$ dimensional matrix that is computed based on the observation sequences (i.e., the temporal order of executions of a system). The probability of each element $b_j(R_k)$ is denoted in equation (2.9) as:

$$b_j(R_k) = P\big(observation\ symbol\ R_k\ at\ t \big| state\ S_j\ at\ t\big) \quad (2.9)$$

Initial State Distribution (π): The third-row stochastic process is the initial state probability distribution $\pi = \{\pi_i\}$. π is a $1\times N$ row matrix and the probability of each element $\{\pi_j\}$ is denoted in equation (2.10) as:

$$\pi_i = P(state\ S_i\ at\ t = 0) \quad (2.10)$$

Training an Ergodic HMM: The behavior of a system can be discrete (e.g., symbols from a finite alphabet) or continuous (e.g., signals from a speech, music, etc.). In our case, the behavior of a process in UNIX or Windows system can be represented as a discrete sequence of system calls. Since a discrete HMM is a stochastic process for sequential data [43] [45], we can use it to learn the behavior of a process. A well-trained HMM model using the discrete normal sequences of system calls can be used as a potential model for detecting anomalies. Practically, training an HMM using a discrete sequence of observation $\mathcal{O}$-$(\mathcal{O}_0, \mathcal{O}_1, \ldots, \mathcal{O}_{T-1})$ aims at maximizing the likelihood function $P(\mathcal{O}|\ \lambda)$ over the parameter space represented by A, B, and π. The Baum-Welch (BW) algorithm is one of the most commonly used Expectation-Maximization (EM) algorithm for learning the HMM parameters [4]. The BW algorithm is an iterative procedure to estimate the HMM parameters. It uses a Forward-Backward (FB) algorithm [45] at each iteration to efficiently evaluate the likelihood function $P(\mathcal{O}|\ \lambda)$, and then updates the model parameters until the likelihood function stops improving or a maximum number of iterations is reached. In our experiments, we have chosen the BW algorithm to train all HMMs using the system calls datasets.

The user-defined three initial distributions of A, B, and π, and two fixed-value parameters of M and N have an impact on the performance of HMM. The common solution for the initial

distributions of A, B and π is the random initialization and the use of validation set to select the best parameters [40]. We have also initialized the distributions of A, B and π randomly and repeated the training process ten times. The initial distributions for which we obtain the highest AUC on the validation set are selected. The alphabet size M is defined by the number of distinct system calls in a system. However, it is difficult to define the number of states N in advance. Because, a single HMM trained with a predefined number of states N may have limited chances to fit the underlying structure of the data [43]. In fact, the underlying distribution of sequences of system calls at different states varies according to the architectural complexity of a system and results in many local maxima of the log-likelihood function [46].

To tackle the variations in the underlying distribution of the sequences of system calls, ensemble HMMs have shown to be a better choice than a single HMM [9] [47]. The ensemble methods have been reported that the diversity among the ensemble classifiers is an essential factor in increasing the accuracy. In particular, Khreich et al., [11] showed that the Iterative Boolean Combination (IBC) of the responses of several accurate and diverse HMM classifiers significantly increase the accuracy while reducing the number of false alarms. We have also trained different discrete-time ergodic HMMs with various N using the BW algorithm. These ergodic HMMs are the primary inputs to the proposed weighted pruning approach for Boolean combination.

2.2.5 One-class Support Vector Machine (OCSVM)

The standard machine learning techniques such as SVM use fixed-size vectors as input features instead of sequential features to model ADS. The bag of system calls, a very effective technique to encode a sequence of system calls into fixed-size vectors [48] [49], adopted from text mining or information retrieval [50] where each unique system call acts as a term or symbol of

alphabet $\sum$ and the number of unique system calls is equal to the size of vectors. We have $m = |\sum|$ unique system calls $\sum = \{v_1, \dots v_m\}$ and a dataset ($\mathcal{T}$) with N labeled sequences $\mathcal{T} = \{< T_i, y_i > | T_i \in \sum^* , y_i \in \{0,1\};\ i = 1, \dots N\}$, where y_i is a corresponding class of labels such that 0 means "*normal*" and 1 means "*anomaly*". Each sequence T_i is then encoded into a term vector $\boldsymbol{\mathcal{V}_i}$ of size m, where each element or system call $v_j \in \sum$ is computed as:

$$\boldsymbol{\mathcal{V}_i}(v_j) = \Phi(v_j, T_i) = \begin{cases} 1, & if\ o_j \in T_i \\ 0, & if\ o_j \notin T_i \end{cases}; \qquad (2.11)$$

$$i = 1, \dots N \ \ and \ \ j = 1, \dots m$$

The term vector $\boldsymbol{\mathcal{V}_i}$ can also be weighted by term frequency (*tf*) as:

$$\boldsymbol{\mathcal{V}_i}(v_j) = \Phi_{tf}(v_j, T_i) = freq(v_j); \quad j = 1, \dots m \qquad (2.12)$$

where $freq$ is the number of times system call v_j appears in T_i, normalized with the length $L = |T_i|$ of sequence T_i. However, Φ_{tf} accounts the discrimination ratio for each term related to only a single sequence. To account for the discrimination ratio for each term over the whole *N* sequences, document frequency (*df*) is proposed. Moreover, the terms that are less frequent across the whole sequences, i.e., the terms with less *df* values are more uncertain, and thus, more informative. Therefore, instead of *df*, they use inverse document frequency (*idf*) as a weighting measure, in order to compute the term vector $\boldsymbol{\mathcal{V}_i}$ as:

$$\boldsymbol{\mathcal{V}_i}(v_j) = \Phi_{idf}(v_j, T_i, \mathcal{T}) = \frac{N}{df(v_j)} freq(v_j); \qquad (2.13)$$

$$\text{j} = 1, \dots m$$

Once a sequential dataset is transformed into a fixed-size (m) vector dataset $\mathcal{T}(T_i, y_i) \rightarrow \mathcal{X}(\boldsymbol{\mathcal{V}_i}, y_i)$ using a weighting function Φ_{tf} or Φ_{idf}. The fixed-size vector-based dataset $\mathcal{X}(\boldsymbol{\mathcal{V}_i}, y_i)$ is then used to train the OCSVM model for anomaly detection.

We use the term vectors $\boldsymbol{\mathcal{V}_i}$, weighted by Φ_{idf} as input features for OCSVM. To train the OCSVM model, we use LIBSVM [51], a library for different types of SVM classifiers. We train the OCSVM using the Gaussian or RBS (radial basis function) kernel function given in Equation (8):

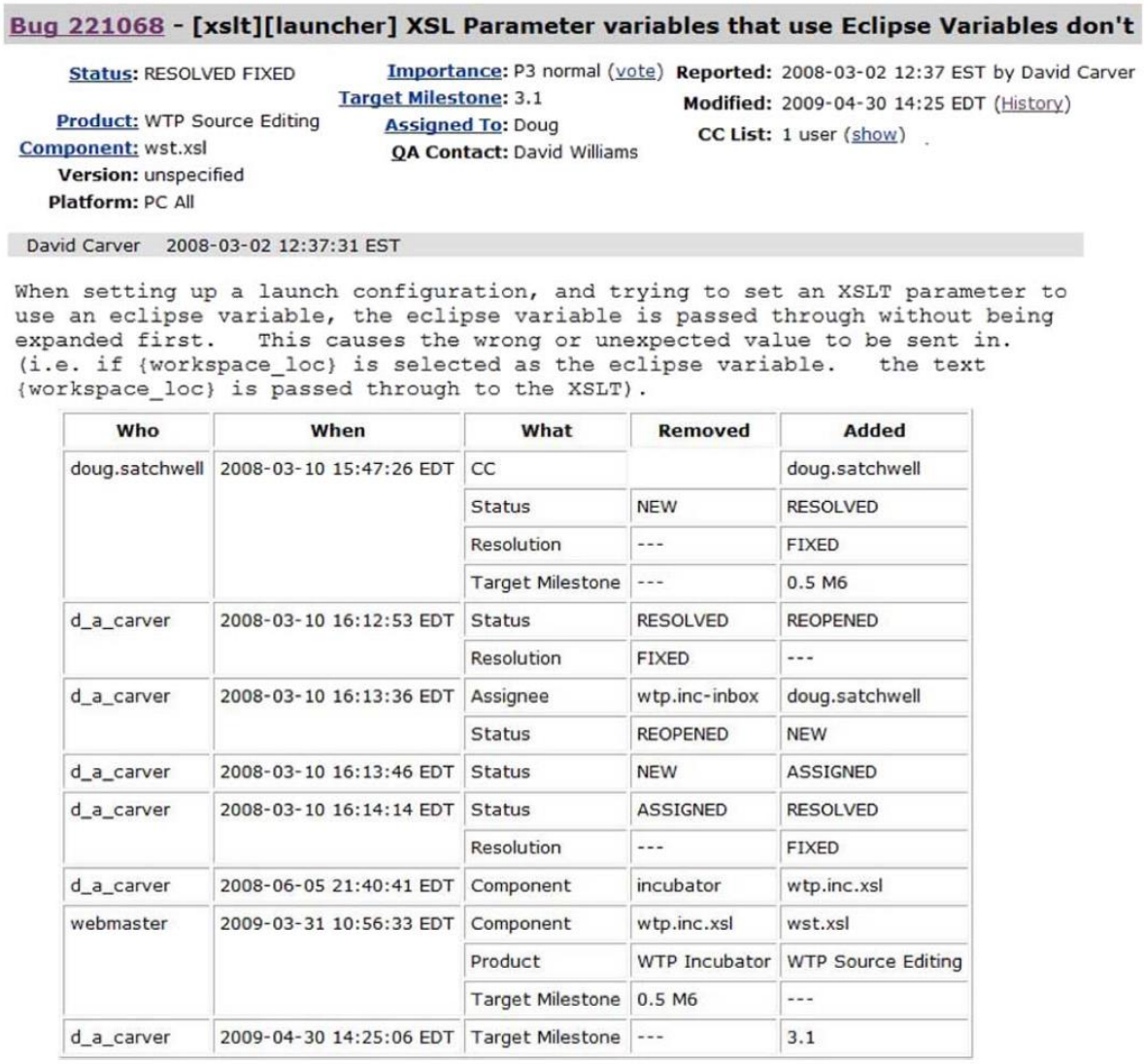
$$\mathrm{K}(v_i, v_j') = \exp\left(-\frac{\|v_i, -v_j'\|^2}{2\sigma^2}\right) \quad i, j = 1, \ldots m \qquad (2.14)$$

2.3 Review of Techniques for Detecting the Reassignments of Bug Report Fields

According to ANSI, the definition of Software Reliability is the probability of bug-free software operation for a specified period of time in a specified environment [52].. We cannot expect a software system with 100% bug free because of the inability to exhaustively test the system. Improving the process of handling bugs by reducing the lead time of fixing bugs contributes to making the system more reliable. In this thesis, we focus on techniques that automatically predict the fields of but reports with the objective of speeding up the bug resolution process.

2.3.1 Reassignments of Bug Report Fields

Bug reports (BRs) contain a wealth of information that is used by triaging and development teams to understand the causes of bugs and provide fixes. The problem is that, for various reasons, it is common to have BRs with missing or incorrect information, hindering the bug resolution process [53] [54] [55]. Xia et al. [56] showed that 80% of the BRs they analyzed (190,558 BRs in total) have their fields reassigned. Figure 2.5 shows an example of a BR (from Eclipse project) with the reassignments of Product, Component, Assignee, and Status fields. Guo et al. [55] argued that the BR field reassignment problem is due to various factors including the difficulty to identify the root cause of a bug, ambiguous ownership of BR components, poor BR quality, difficulty to determine the proper fix, and workload balancing.



Bug 221068 - [xslt][launcher] XSL Parameter variables that use Eclipse Variables don't

Status: RESOLVED FIXED
Importance: P3 normal (vote)
Reported: 2008-03-02 12:37 EST by David Carver
Target Milestone: 3.1
Modified: 2009-04-30 14:25 EDT (History)
Product: WTP Source Editing
Assigned To: Doug
CC List: 1 user (show)
Component: wst.xsl
QA Contact: David Williams
Version: unspecified
Platform: PC All

David Carver 2008-03-02 12:37:31 EST

When setting up a launch configuration, and trying to set an XSLT parameter to use an eclipse variable, the eclipse variable is passed through without being expanded first. This causes the wrong or unexpected value to be sent in. (i.e. if {workspace_loc} is selected as the eclipse variable. the text {workspace_loc} is passed through to the XSLT).

Who	When	What	Removed	Added
doug.satchwell	2008-03-10 15:47:26 EDT	CC		doug.satchwell
		Status	NEW	RESOLVED
		Resolution	---	FIXED
		Target Milestone	---	0.5 M6
d_a_carver	2008-03-10 16:12:53 EDT	Status	RESOLVED	REOPENED
		Resolution	FIXED	---
d_a_carver	2008-03-10 16:13:36 EDT	Assignee	wtp.inc-inbox	doug.satchwell
		Status	REOPENED	NEW
d_a_carver	2008-03-10 16:13:46 EDT	Status	NEW	ASSIGNED
d_a_carver	2008-03-10 16:14:14 EDT	Status	ASSIGNED	RESOLVED
		Resolution	---	FIXED
d_a_carver	2008-06-05 21:40:41 EDT	Component	incubator	wtp.inc.xsl
webmaster	2009-03-31 10:56:33 EDT	Component	wtp.inc.xsl	wst.xsl
		Product	WTP Incubator	WTP Source Editing
		Target Milestone	0.5 M6	---
d_a_carver	2009-04-30 14:25:06 EDT	Target Milestone	---	3.1

Figure 2.5. Reassigned and refined Bug Report of Eclipse Project with BugID 221068 [56].

2.3.2 Background

To address the BR field reassignment problem, researchers (e.g., [53] [56] [19]) have turned to machine learning techniques. The common practice is to design predictive models that leverage historical BRs (the ground truth) to automatically predict whether a field of an incoming BR would most likely get reassigned or not. Existing approaches rely mainly on traditional classification algorithms such as SVM, KNN, decision trees, and the combination of these. For example, Xia et

al. [56] trained a multi-label imbalanced KNN model (Im-ML.KNN) that combines three multi-label classifiers built using BR field metadata, BR descriptions and summaries, and a mix of both. Other methods include the use Naïve Bayes [53], ML.KNN [57], and HOMER [58].

Although these approaches have been shown to be successful at varying degrees, they do not take full advantage of the sequential order of information in BR data such as function call sequences in stack traces, which may lead to improved prediction accuracy. To enable the modeling of sequential data, in Chapter 4, we propose an approach, called EnHMM, which leverages the power of HMMs to predict the reassignment of BR fields. An HMM is a classification technique (more precisely a stochastic process) that is designed specifically to model sequential data [59]. HMMs are widely used in other areas such as intrusion detection [15], DNA processing [60], speech recognition [44], and image processing [61]. EnHMM combines multiple HMMs (trained by varying the number of hidden states) using our proposed WPIBC Boolean combination technique [17]. This design choice is inspired by studies in the field of anomaly detection (e.g., [15] [24] [11]), which showed evidence that the combination of multiple HMMs increases accuracy over a single HMM.

We use stack traces as the main features for our EnHMM approach. A stack trace contains a sequence of function calls that are in memory when a bug occurs, which we believe is a better characterization of the bug as opposed to BR descriptions, entered by end users. We conjecture that a best-fit ensemble HMM model, trained on stack traces of reassigned and not reassigned BRs, would help predict the probability of an unknown BR field.

2.3.3 Related Work

The closest work to our study is that of Xia et al. [56]. They built a model to predict

reassignment of BR fields using multi-label learning algorithm (ML.KNN). Their method (im-ML.KNN) combines three different classifiers based on BR field metadata, BR descriptions and summaries, and a combination of these features. Their approach achieved an accuracy (F-measure) ranging from 56% to 62%.

Bettenburg et al. [54] conducted a survey among developers and users of Apache, Eclipse, and Mozilla to understand what makes a good BR. They showed that since users are not primarily technical domain experts, they cannot choose BR fields correctly. They found that the steps to reproduce and stack traces are the most useful fields in BRs. Incomplete information in BRs appears to be one of the problems encountered by developers to fix the bugs.

Guo et al. [55] [62] showed that there are five main reasons for BR field reassignment: Finding the root cause, determining ownership, identifying the root cause (proper fix determination), poor BR quality (incorrect or incomplete BRs), and workload balance. They showed that imprecise BR fields lead to the BR being transferred between development teams. They referred to this fact as the bug pong concept. They also showed that the incorrect selection of BR fields, increases the bug fixing time. Breu et al. [62] [63] showed that BR questions can be categorized into eight groups: Missing information, clarification of information provided, information for triaging, information needed for debugging, information on how to provide corrections, status inquiry, resolution, and administration questions. They also showed that incorrect information is the main cause of triaging uncertainties.

Shihab et al. [63] [64] showed that BRs that are reassigned take in average two times longer to be fixed. Sureka [65] showed that the Assignee field is the most reassigned field in the bug repositories. He applied a probabilistic model to the title and description of BRs to predict faulty

component fields. The approach could be used to predict faulty component field of BRs with 42% accuracy. Lamkanfi et al. [53] showed that faulty component field of Eclipse and Mozilla BRs are frequently reassigned. They trained a Naïve base classifier to predict reassignment of the component field of BRs in Eclipse and Mozilla based on BR component, reporter, operating system, version, severity, and summary. They showed that their approach achieves an accuracy of 44% for predicting if a bug will be reassigned and 83% if a bug will not be reassigned.

Several studies focused on using stack traces to detect duplicate BRs [19] [66] [67]. These studies build feature vectors based on the functions in stack traces. They showed that predictive models built based on stack traces can detect duplicate BRs with an accuracy of up to 90%. Other studies focused on using stack traces to predict BR fields. Sabor et al. [68] [69] [70] [71] built feature vectors based on the functions in stack traces. They showed that traces and BR categorical feature provide good accuracy.

2.4 Review on Detecting System Anomalies Using Big Data Platform

2.4.1 MapReduce Programming Model and Hodoop

The MapReduce programming model uses split-apply-combine strategy for processing and generating Big Data with commodity hardware [72] [73]. A MapReduce job is composed of two functions: Mapper and Reducer. The Mapper function reads each line of record from an input file, performs some operations, and produces a list of key-value pairs as output. The Reducer function takes all the intermediate values associated with a particular key, applies defined actions, and writes the results into the output files. Both Mapper and Reducer functions are designed to run simultaneously and independently on each node in a cluster.

Apache Hadoop [74] implements the MapReduce programming model with the distributed

file system, known as Hadoop Distributed File System (HDFS). Hadoop splits a file into large blocks (typically, 64MB) and distributes them across several parallel nodes. Each node only accesses and processes the assigned data locally, which yields greater efficiency [75]. Moreover, Hadoop is scalable, fault tolerant, cost effective and flexible. As a result, it has become the industry standard for handling Big Data. A small Hadoop cluster has one master and multiple worker nodes. The master node contains JobTracker, TaskTracker, Name Node, and Data Node whereas the slave or worker node contains only TaskTracker and Data Node. The JobTracker initializes a MapReduce job and manages the TaskTracker on each node. The TaskTracker on each node executes the Mapper and Reducer tasks assigned by the JobTracker.

2.4.2 Background

Most reported approaches [15] [11] [48] [49] for anomaly detection were based on sequence matching. During training, these approaches built the normal profile by filtering and transforming the large-scale traces of system calls into numerical sequences of system calls, and then, treating them to profile heterogeneous features for heterogeneous anomaly classifiers. For example, OCSVM [48] [49] uses the fixed-size vector-based features while IBC [11] and WPIBC [15] use fixed-size sliding window-based short sequences of system calls.

Therefore, the very first step for any ensemble of heterogeneous anomaly classifiers is to profile the heterogeneous features for the heterogeneous anomaly classifiers. Two important tasks are needed to profile the heterogeneous features: preprocess the large-scale traces of semi-structured data and then, extract the target features from that preprocessed sequential data. However, it is difficult for a single machine to handle such huge compute incentive tasks. So far, the smart solution is leveraging the power of existing parallel computation frameworks, such as

HDFS (Hadoop Distributed File System) and the MapReduce programming model.

However, Hadoop with its original parallel computation model is technically not suitable for profiling sequential data due to dependencies on the temporal information or the orders of a sequence [76]. For example, when HDFS splits a large trace file into two or more fixed-size blocks, Hadoop fails to keep track of the order or temporal information of large sequences within the trace file. Therefore, we need a MapReduce solution that profile the heterogeneous features such as fixed-size sliding windows for short sequences-based anomaly classifiers (e.g., HMMs and STIDE) and fixed-size feature vectors for the traditional machine learning based anomaly classifiers (e.g., OCSVM).

2.4.3 Related Work

Several studies have been proposed in the literature to deal with the system anomalies detection problem using Big Data platforms, particularly, Hadoop and MapReduce programming model [76] [77] [78] [79]. Among them, Matthews et al. [77] have recently proposed a MapReduce solution for detecting real-time anomalous behaviors in SCADA systems. They analyzed both the voltage and current phasors, as well as a set of frequency measurements to detect any deviations from the true value. However, this solution is technically not suitable for utilizing the power of MapReduce and Hadoop to profile short sub-sequences or time slice windows from a large-scale temporal data. This is due to the fact that the latter assume that the data should be preprocessed and stored in a CSV file before being used. Moreover, traditional machine learning approaches [80] [81], use fixed-size feature vector instead of short sub-sequences. Therefore, this solution [77] is suitable for a single-based anomaly classifier with a preprocessed time slice data and not appropriate for ensemble-based anomaly detection systems.

Zhenlong Li et al., [78] proposed a spatiotemporal indexing approach that can be used by a MapReduce job for retrieving and processing spatiotemporal climate data. They used the proposed index data structure as a global grid, which is accessed by each node for re-assembling the features from a block of data. However, the size of the global indexing grid increases exponentially with the increase of the spatiotemporal resolution (or time slice) size. Therefore, the spatiotemporal indexing is reliable when the time slice is large (e.g., daily basis). For a small window, however, the size of each global grid may reach several gigabytes which reduces the computational efficiency.

Kim et al., [79] proposed a host-based anomaly detection method by leveraging the Hadoop MapReduce parallel computation model in the era of host-generated Big Data. They reported that the behavior of malicious codes is logged basically on the host. They analyze the host log information which includes various log data such as enormous amounts of security logs, network and host information, and application transactions. This approach is also limited to profile only vector-based features. In that case, our proposed MapReduce solution, MASKED takes a full of advantage of the parallel computation framework, Hadoop, by profiling heterogeneous features and processing them using a pre-constructed ensemble-based BICKER Boolean combination rules.

Chapter 3. Anomaly Detection Techniques Based on Weighted Kappa-Pruned Ensemble of HMMs

In this chapter, we propose weighted pruning based Boolean combination, an efficient approach for selecting and combining accurate and diverse anomaly classifiers. It works in three phases. The first phase selects a subset of the available base diverse soft classifiers by pruning all the redundant soft classifiers based on a weighted version of Cohen's kappa measure of agreement. The second phase selects a subset of diverse and accurate crisp classifiers from the base soft classifiers (selected in Phase1) based on the unweighted kappa measure. The selected complementary crisp classifiers are then combined in the final phase using Boolean combinations. The results on two large scale datasets show that the proposed weighted pruning approach is able to maintain and even improve the accuracy of existing Boolean combination techniques, while significantly reducing the combination time and the number of classifiers selected for combination.

3.1 Introduction

In this work, we propose a weighted pruning of Boolean combinations that selects the best subset of diverse base soft classifiers by pruning all the redundant ones. Each diverse base soft classifier is then used independently to select the complementary crisp classifiers instead of brute-force search like in PBC. The complementary crisp classifiers are then combined by leveraging both Pair-wise Brute-force Boolean Combination (BBC2) and Iterative Boolean Combination (IBC) [10] [11], which shows that the pruning approach can be used with any Boolean combination approach.

We leverage both weighted and unweighted Cohen's kappa [82] [83] in order to select the best subset of diverse base soft classifiers. Weighted Cohen's kappa is a special case of simple

kappa (unweighted kappa) that is particularly used when the agreements between two classifiers are ordinal instead of nominal. In our case, the scores of a soft classifier are ordinal and the decision of a crisp classifier based on a given threshold is nominal. Our weighted pruning approach prunes both soft and crisp classifiers based on the ordinal agreements and the nominal agreements between two classifiers. The selected diverse and accurate crisp classifiers are then used for Boolean combination. During combination, we leverage both the pair-wise and iterative Boolean combinations introduced by Barreno et al. [10] and Khreich et al. [11], respectively. The proposed Pair-wise Weighted Pruning Boolean Combination (namely called WPBC2) fuses and combines all possible pairs of crisp classifiers generated from the selected diverse base soft classifiers. Whereas, the Weighted Pruning Iterative Boolean Combination (namely called WPIBC) fuses and combines the selected diverse base soft classifiers sequentially until no significant improvement is possible. Another major contribution of this paper is the evaluation of our approach for detecting anomalies at the system call levels. We compare the performance of WPBC2 and WPIBC to that achieved with the original BBC2 and IBC techniques. In addition, we compare the performance of our approaches to Pruning Boolean Combination (PBC) [12].

The main contributions of this work are:

1. We propose an anomaly detection approach that enforces the diversities among the combined soft and crisp classifiers using weighted and unweighted Cohen's kappa [82].

2. The approach can be used with both pair-wise and iterative Boolean combination techniques [10] [11], and easily adaptable to other Boolean combination methods.

3. We evaluate our approach on two large publicly available system call datasets: ADFA Linux Dataset (ADFA-LD) [7] and CANALI Windows Dataset (CANALI-WD) [84].

4. We show that our approach outperforms BBC2, IBC, and PBC by achieving lower false positive rate, while maintaining and improving the detection accuracy, measured using AUC.

3.2 Proposed Weighted Pruning Technique

The proposed weighted pruning based Boolean combination approach leverages both weighted and unweighted kappa measures of (dis)agreement. The main novelty of this work is to ensure that the diversity among the scores of all the available ensemble of soft classifiers by pruning the redundant soft classifiers using weighted kappa. Then, our approach applies the unweighted kappa based MinMax-Kappa pruning technique (one of the pruning techniques of PBC) individually on each selected diverse base soft classifiers and selects the complementary crisp classifiers. At the end, we merge all the selected complementary crisp classifiers from each selected diverse base soft classifiers and use them for Boolean combination.

3.2.1 Kappa Measure of (Dis)Agreement

Cohen's kappa or simply called kappa is a statistical tool that is widely used for measuring the inter-rater reliability or (dis)agreement between raters [6]. There are two types of kappa coefficients that can be used in computing the inter-rater reliability. The unweighted kappa coefficient is the simplest version of kappa [83] that is used only for nominal category. The weighted kappa coefficient is an extended version of kappa [83] that is used when the category is ordinal [85]. Our pruning techniques leverage both kappa coefficients. The weighted kappa coefficient is used to prune the redundant soft classifiers when the level of scores is ordinal (thresholds). And the unweighted kappa coefficient is used to prune the trivial and redundant crisp classifiers when the decision is nominal (anomaly/normal).

The contingency matrix for both kappa coefficients of (dis)agreement is defined on two

classifiers. Let the two classifiers be *D1* and *D2* and the contingency matrix is $C_{n\times n}$. Here, *n* is the order of levels. For unweighted kappa coefficient, *n* is fixed to two that is either positive or negative. For weighted kappa coefficient, *n* is equal to the number of levels or thresholds with the assumption that both classifiers have the same number of constant levels or thresholds. An example of a contingency table $C_{2\times 2}$ for *n=2*, is given in Table I. Where, each element a_{ij} represents the number of instances on which classifier *D1* and classifier *D2* agree at $level_i$ and $level_j$. The sum of all elements in Table I is equal to the size of the validation set.

Table I: Contingency Matrix

		D1	
		Positive/$level_1$	Negative/$level_2$
D2	Positive/$level_1$	a_{11}	a_{12}
	Negative/$level_2$	a_{21}	a_{22}

For the weighted kappa coefficient, we need to define the weighted matrix W in addition to the contingency matrix C. Among the many possible weighting schemes, the linear weighting scheme is effective when one order is important than the next one [60]. We also use linear weight when the order is the number of thresholds and the distance between two thresholds is important to define whether two soft classifiers are similar or diverse. We can compute the linear weighting matrix W using equation (3.1).

$$W = w_{ij} = 1 - \frac{abs(i-j)}{n-1} \tag{3.1}$$

When C and W are the same dimensional square matrices, the kappa coefficient for both unweighted and weighted kappa can be computed based on the Hadamard product (o) [82] or element-wise product of matrices according to the equation (3.2):

$$kp = \frac{p_a - p_\varepsilon}{1 - p_\varepsilon} \tag{3.2}$$

where $p_a = sum(CoW)$ is the proportion of weighted agreement (for unweighted kappa, $W = I$ means complete agreement). The parameter p_ε is the proportion of agreement due to chance and computed using equation (3.3) as:

$$p_\varepsilon = (c_{n\times 1} \times r_{1\times n})oW \tag{3.3}$$

(a)

S(D1)- 3.9|4.3|3.4|3.2|3.6|3.1|3.0|3.0|2.3|2.6|2.5|2.2|0.8|1.4|0.8|1.9|0.8|1.1|1.3|0.8|0.8|0.8|1.1|10.5|0.8
S(D2)- 3.2|3.4|3.1|3.0|3.7|3.0|3.0|3.0|2.3|2.5|2.5|2.2|0.9|1.2|0.9|1.8|0.9|1.0|1.2|0.9|0.9|0.9|1.3|10.8|0.9
S(D3)- 2.9|2.8|3.2|3.0|3.1|2.9|2.8|2.9|2.2|2.5|2.4|2.3|1.9|2.7|1.9|1.9|1.9|2.4|2.0|1.9|1.9|1.9|2.4|11.2|1.9

(b)

Testing soft detectors D2 and D3	Base soft detector (D1)			
Levels	$S \geq 3$	$3 > S \geq 2$	$2 > S \geq 1$	$1 > S \geq 0$
$S \geq 3$	9_4_1.00	0_0_0.66	0_0_0.33	0_0_0.00
$3 > S \geq 2$	0_5_0.66	4_4_1.00	0_4_0.66	0_0_0.33
$2 > S \geq 1$	0_0_0.33	0_0_0.66	5_1_1.00	0_7_0.66
$1 > S \geq 0$	0_0_0.00	0_0_0.33	0_0_0.66	7_0_1.00

(c)

Ground Truth-11111000000000000000000000	Ground Truth-11111000000000000000000000
D1: R(S ≥ 3)- 11111111000000000000000010 D2: R(S ≥ 3)- 11111111000000000000000010 D3: R(S ≥ 3)- 00111000000000000000000010	D1: R(S ≥ 2)- 11111111111000000000010 D2: R(S ≥ 2)- 11111111111000000000010 D3: R(S ≥ 2)- 11111111111010001100010
D1: R(S ≥ 1)- 11111111111010101100010 D2: R(S ≥ 1)- 11111111111010101100010 D3: R(S ≥ 1)- 11111111111111111111111	D1: R(S ≥ 0)- 11111111111111111111111 D2: R(S ≥ 0)- 11111111111111111111111 D3: R(S ≥ 0)- 11111111111111111111111

(d)

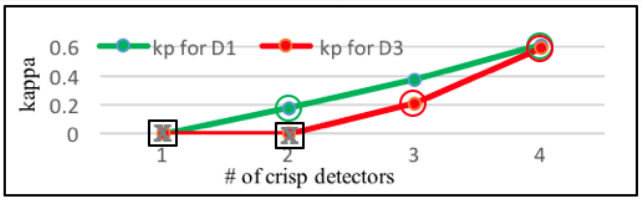


Figure 3.1. A simple example of weighted and unweighted kappa for pruning redundant soft and crisp classifiers

Here, $c_{n\times 1}$ denotes a column matrix in which each element is the sum of each row of C. Similarly, $r_{1\times n}$ is a row matrix in which each element is the sum of each column of C. The kappa coefficient kp computes the inter-rater reliability based on the proportion of agreement (p_a) and

agreement due to chance (p_ε), where the degrees of disagreement are controlled by the weight matrix W ($W = I$ for unweighted kappa that means no degrees of disagreement). Therefore, $kp = 1$ indicates perfect agreement (i.e., both classifiers agree at the same level for every instances) and $kp = 0$ indicates that any agreement is totally due to chance. The value of kp might also be negative. Negative values indicate both classifiers are negatively correlated, and such complementary classifiers are important in the combination of ensemble techniques [13] [14].

In the rest of this work, we use the running example shown in Figure 3.1 to describe the phases of our approach. In this example, we have selected three HMM-based classifiers, D_1, D_2, and D_3 by varying the number of hidden states. Figure 3.1 (a) shows the scores of each classifier.

Phase1-Pruning Using Weighted Kappa: The first phase of Algorithm 1 describes the steps for pruning the redundant soft classifiers using weighted kappa coefficient kp. Suppose, we have K soft classifiers and they produce $S_k\{k = 1 \dots K\}$ score vectors using a validation set V. In the example of Figure 3.1, K = 3 and the scores for each classifier are shown in Figure 3.1 (a). Let the number of thresholds of each soft classifier be n_k. In the example of Figure 3.1, $n_k = 4$. Therefore, we have *K ROC* curves (S_k, n_k) with probably *K* different *AUC* values. In each iteration (lines 7-18 in Algorithm 1), we select one out of *K* available soft classifiers for which the *AUC* is maximum and use it as a base soft classifier S_b. We store S_b onto B (line 9 in Algorithm 1) for the next Phase2. Now, we compute the weighted kappa coefficients kp between S_b and each of the rest $K \leftarrow K - S_b$ soft classifiers where the thresholds n_k of S_b are used as an order or levels. Then, the soft classifiers among the $K - S_b$ soft classifiers which perfectly agree $(0.8 < kp \leq 1)$ with S_b based on the computed weighted kappa *kp*, are pruned as a redundant copy of S_b. Let say, the number of redundant classifiers we found in each iteration is $0 \leq K' \leq K - 1$, and then we remove

them from the available K classifiers as: $K \leftarrow K - K'$. We repeat this process until K is zero.

Algorithm 1: $PSCDs(S_1, \ldots S_K, T_1, \ldots T_K, lab)$: Pruning Soft and Crisp Classifiers

input: scores of K soft classifiers $\{S_1, \ldots S_K\}$ on a validation set along with their thresholds $\{T_1, \ldots T_K\}$, and true labels lab of size $|lab|$.
output: selected $L \ll K$ diverse base soft classifiers $\{B_1, \ldots B_L\}$ along with their complementary crisp classifiers or thresholds $\{\theta_1, \ldots \theta_L\}$ where $\theta_l \ll T_l$ ($\theta_l = 12$ and $T_l = 100$ on average)

```
// Phase1-pruning soft classifiers using weighted kappa
allocate an array AUC_all[1:K]   // temporary store auc of each S_k
for k ← 1 to K do
        compute auc of ROC(S_k,T_k)
        push auc onto AUC_all
allocate an empty array B = []            //store selected diverse soft classifiers
while (K)
        select base soft classifier: S_b ← max_k[AUC_all(k)]
        store S_b onto B   // store S_b as a base soft classifier
        let n_b ← number of order/levels/thresholds in T_b
        update K ← K - S_b                    // remove S_b from K soft classifiers
        update AUC_all ← AUC_all - AUC_all(S_b)          // remove auc for S_b
        let n ← the size of | K |
        for k ← 1 to n do
                compute linear weighted kappa kp between S_k and S_b using n_b
                if  0.80 < kp <=1
                        update K ← K - S_k   // remove S_k as a redundant copy of S_b
                        update AUC_all ← AUC_all - AUC_all(S_k)    // remove auc for S_k
// -----Phase2- pruning crisp classifiers using unweighted kappa----------
let L ← number of selected diverse base soft classifiers in B
let m ← number of selected complementary crisp classifiers from S_b ∈ B
allocate an empty array θ = []  //store thresholds of each complementary crisp     //classifiers
for b ← 1 to L do
        let n_b ← number of crisp classifiers or thresholds in T_b ∈ S_b
        allocate an array U[1: n_b]    // store temporary kappa coefficients
        allocate an array V[|lab|: n_b] //store temporary responses
        for j ← 1 to n_b do
                r ← S_b ≥ t_j  //temporary responses at decision threshold t_j ∈ T_b
                compute unweighted kappa kp between r and lab
                push kp onto U and r onto V
        filter U and V by removing trivial classifiers
        select m complementary crisp classifiers using MinMaxKappa(U,V) pruning technique
        map m selected complementary crisp classifiers into θ_b thresholds
        store θ_b thresholds onto θ// store θ_b complementary crisp classifiers of S_b
return  B < S_1, ... S_L > and θ < θ_1, ..., θ_L >
```

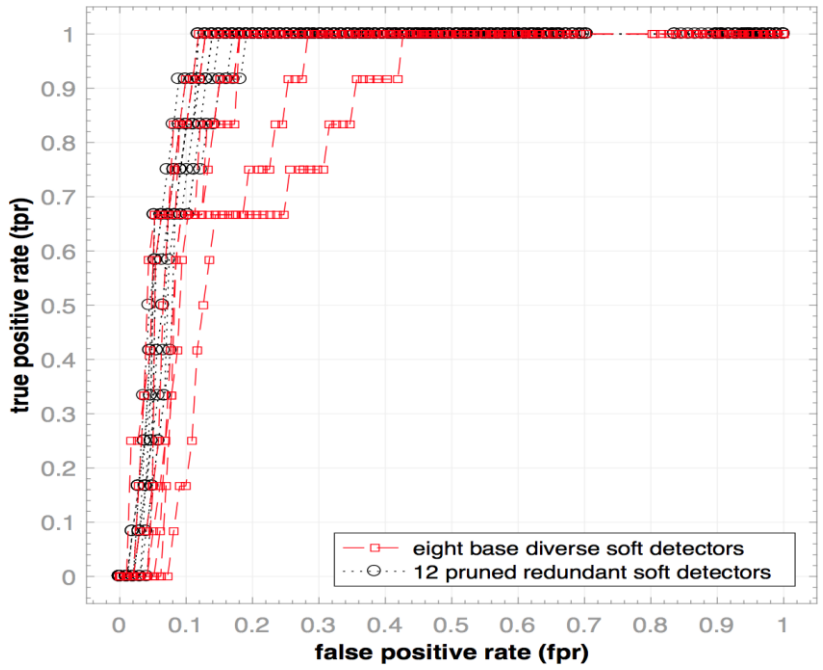
Using the example shown in Figure 3.1, we have $S_b = D_1$ because the AUC of D_1 is maximum. We then store D_1in B as a base soft classifier. Suppose, n_k of $D1$ is equal to four different levels ($S \geq 3;\ 3 > S \geq 2; 2 > S \geq 1;\ and\ 1 > S \geq 0$) of scores $S(D_1)$. First, we have to compute the contingency and weighted matrices between base ($S_b = D_1$) and each of the rest two ($K \leftarrow K - S_b$) soft classifiers D_2 and D_3. Figure 3.1 (b) shows the contingency tables ($C_{4\times4}$)

for four different levels. Since the dimension of the contingency and weighted matrices are the same, we put them together, where, each cell $c_{ij}(\#\_\#\_w_{ij})$ in Figure 3.1 (b) represents three values: The first and second values represent the number of samples agreed at levels i and j of the two contingency tables between D_1 and D_2 and between D_1 and $D_{3,}$ respectively. The third value is the linear weight, computed using Equation (3.1).

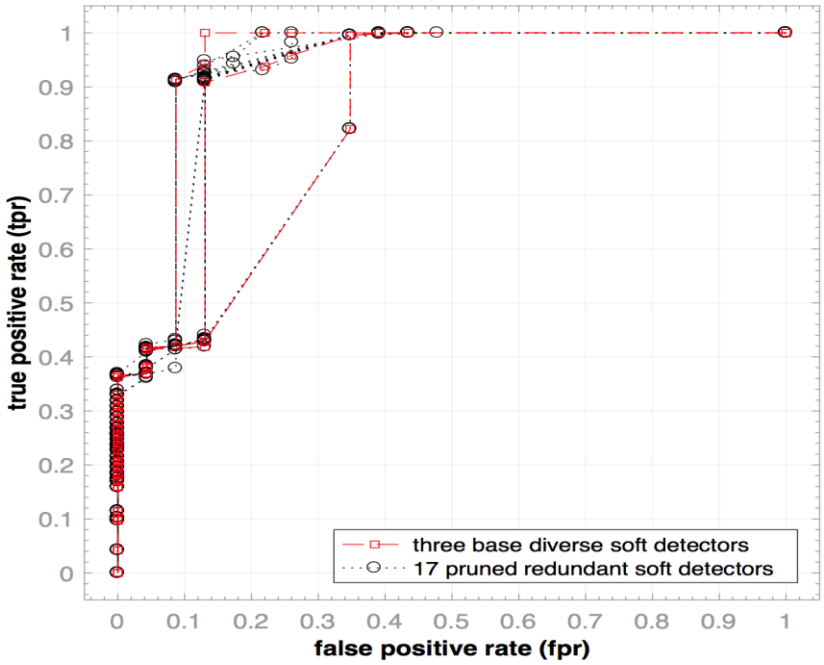
Based on the contingency and weighted matrices between two classifiers, we can compute the weighted kappa (kp) coefficients using Equation (3.2). The weighted kappa kp between D_1 and D_2 is 1, meaning that both are in perfect agreement (i.e., $kp \in 0.8 < kp \leq 1$) at the same level for every instance, and thus D_2 should be pruned (lines 15 to 18 in Algorithm 1). However, the weighted kappa kp between D_1 and D_3 is 45.65, meaning poor agreement (i.e., $kp \notin 0.8 < kp \leq 1$) at the same level for every instance, and therefore D_3 is more likely to diverse from D_1 and should be selected for combination. At the end of the first iteration, we only keep D_3 (i.e., K=1), while D_2 is pruned because it is redundant of the base classifier, D_1. The final results of this phase consist of two diverse base soft classifiers D_1 and D_3. The diversities at the response level for four different thresholds are presented in Figure 3.1 (c). We can see that the responses of the two selected base soft classifiers, D_1 and D_3, diverse at various instances (see Figure 3.1 (c)) for all threshold points, except for $S \geq 0$.

In Figure 3.2 (a), we show a more realistic example, using the ADFA-LD dataset with 20 soft HMM classifiers. In this figure, we have eight base soft diverse classifiers (green solid ROC curves) and 12 pruned redundant soft classifiers (black dotted ROC curves). Similarly, Figure 3.2 (b) shows the experiment on CANALI-WD dataset, where we have only three base soft diverse classifiers and 17 pruned redundant soft classifiers. At the end of Phase1, all the selected base soft

diverse classifiers $L \ll K$ (stored in B) are then fed into Phase 2 of Algorithm 1.

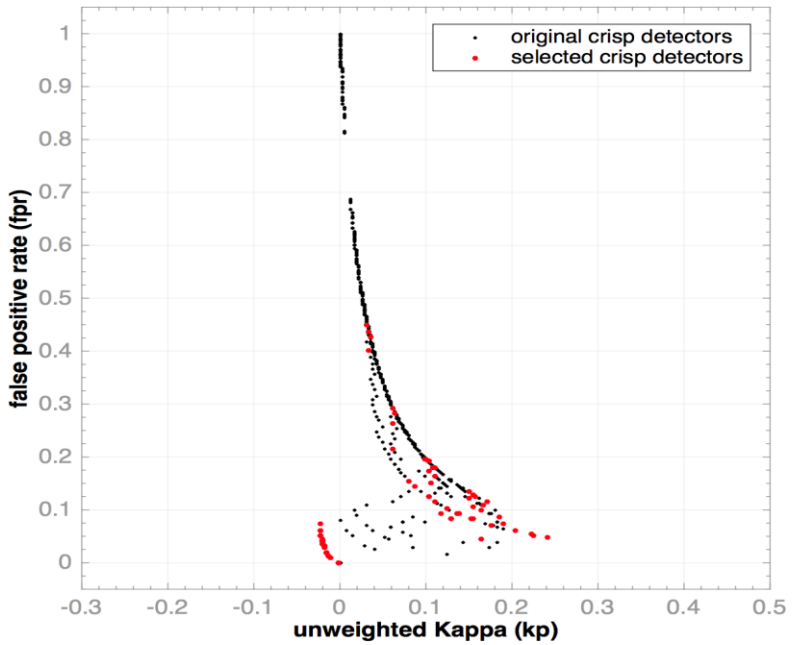


(a) Diverse and redundant soft classifiers on ADFA-LD dataset

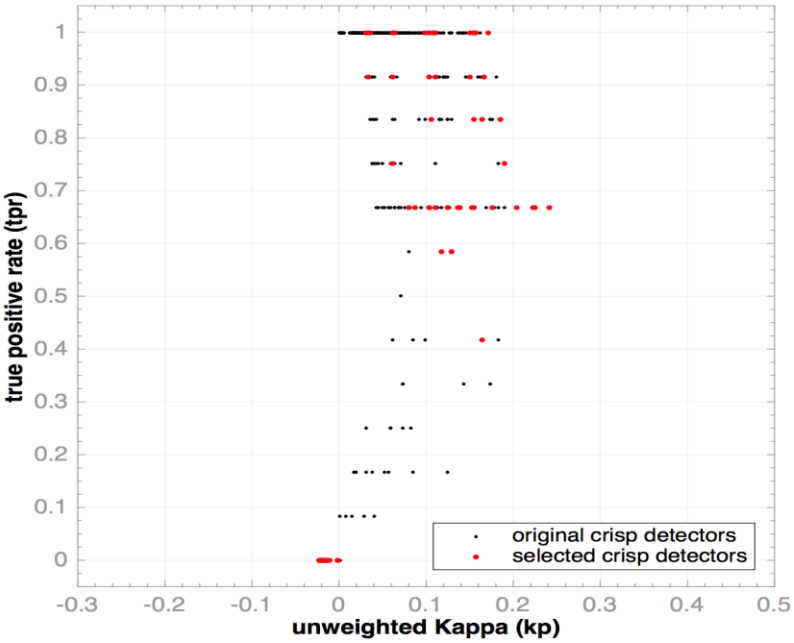


(b) Diverse and redundant soft classifiers on CANALI-WD dataset

Figure 3.2. Example of selected base soft classifiers (green solid lines) with pruning redundant soft classifiers (doted black lines) under the ROC space using weighted kappa (*Phase1* in Algorithm 1) on ADFA-LD dataset (a) and CANALI-WD dataset (b).



(a) *kp-fpr* diagram



(b) *kp-tpr* diagram

Figure 3.3. Example of selected complementary crisp classifiers (red bold points) under the simple kappa versus true positive rate (kp-tpr) diagram (a) and kappa versus false positive rate (kp-fpr) diagram (b) with pruning trivial and redundant crisp classifiers (small black points) from the *L* base soft classifiers (selected by *Phase1* in Algorithm 1) using *MinMax-Kappa* pruning technique (*Phase2* in Algorithm 1) on ADFA-LD dataset.

***Phase2-Pruning Using Unweighted Kappa*:** The second phase of Algorithm 1 leverages the *MinMax-Kappa* pruning method [12], one of the two pruning methods of PBC using unweighted kappa, to select the complementary crisp classifiers. Since the base soft classifiers selected in Phase1 are diverse, we apply the *MinMax-Kappa* pruning method on each base soft classifier individually instead of brute-force search like in PBC. We compute the unweighted kappa coefficient kp between a base soft classifier's decision vector (or crisp classifier) and the true decision labels (or ground truth), same as in PBC. If n_b is the number of decision levels on a base classifier's scores vector S_b, then we obtain n_b crisp classifiers. Now, we compute unweighted kappa coefficients of n_b crisp classifiers and sorted them in ascending order. According to *MinMax-Kappa*, the accurate crisp classifiers should reside close to $kp \approx kp_{max}$ and their complementary crisp classifiers should reside close to $kp \approx kp_{min}$. However, we have to set the number of crisp classifiers and the ratio of them to be selected close to kp_{max} and kp_{min}. We set the ratio is 50%, same as in *MinMax-Kappa*. Moreover, before selecting the complementary crisp classifiers, we have to filter out the trivial crisp classifiers (giving always either positive or negative responses) whose kp is close to zero.

In the running example shown in Figure 3.1, Phase2 selects two diverse base soft classifiers D_1 and D_3 with four different thresholds. Therefore, each base soft classifier produces four crisp classifiers at four different levels or thresholds. The responses $R(D(S) \geq \theta)$ of each crisp classifier for 25 instances and their corresponding true labels (ground truth) are shown in Figure 3.1(c). Figure 3.1(d) shows the unweighted kappa values sorted in ascending order for each crisp classifier of two base soft classifiers D_1 and D_3.

Consider a ratio of 50% and the number of crisp classifiers to be selected to be two. Therefore,

from Figure 3.1(d), we obtain, $kp_{max} \approx 0.62$ and $kp_{min} \approx 0$ for D_1. Similarly, for D_3, $kp_{max} \approx 0.59$ and $kp_{min} \approx 0$. However, the trivial crisp classifiers, one for D_1: $R(S \geq 0)$; and two for D_3: $R(S \geq 1)$ & $R(S \geq 0)$ should be filtered out first. Figure 3.1(d) shows the filtered trivial crisp classifiers (large diagonal marker with cross sign). Since the ratio is 50%, from each base soft classifier, one crisp classifier should be selected close to kp_{max} and another one should be selected close to kp_{min}. Figure 3.1(d) shows the four selected complementary crisp classifiers (two from each base soft classifier, marked with large circle marker).

In general, if the number of selected complementary crisp classifiers from a selected base soft classifier is *m* (i.e., *m/2* close to kp_{max} and *m/2* close to kp_{min}), then the total number of selected crisp classifiers will be $M = m * L$, where L is number of selected base soft classifiers (selected from Phase1). We tested *m* with different setting (*l=4, 8, 12, 16, and 20*) and obtained best results for *m = 12*.

In Figure 3.3, we show a more realistic example, using ADFA-LD dataset. In this figure, we have M complementary crisp classifiers (red bold points) selected from L diverse base soft classifiers (selected in Phase1) using unweighted kappa-based *MinMax-Kappa* pruning technique. Figure 3.3 (a) shows the results under the space of *kp-fpr* and Figure 3.3 (b) shows the results under the space of *kp-tpr*. Figure 3.4 also shows the selected total $M = 96$ complementary crisp classifiers from the $L = 8$ diverse base soft classifiers under the ROC space.

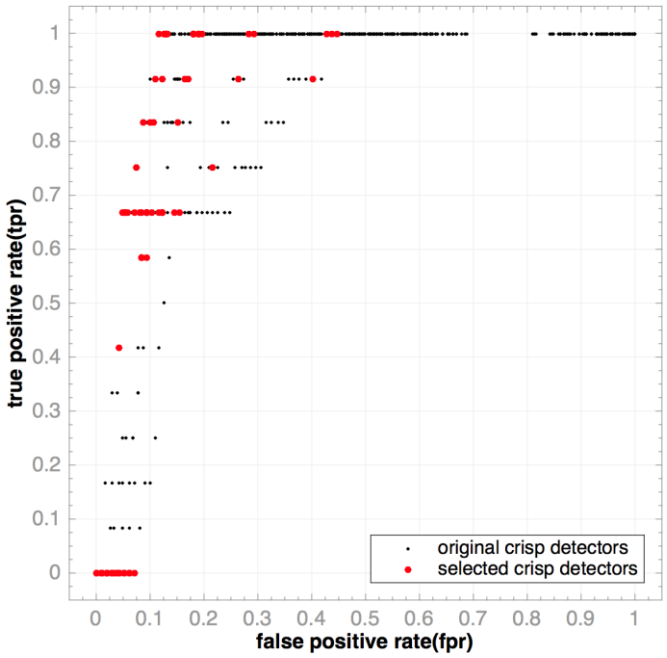


Figure 3.4. Example of selected complementary crisp classifiers (red bold points) under the ROC space with pruning trivial and redundant crisp classifiers (small black points) from the *L* base soft classifiers (selected by *Phase1* in Algorithm 1) using *MinMax-Kappa* pruning technique (*Phase2* in Algorithm 1) on ADFA-LD dataset

Phase3-Boolean Combination Techniques: The third phase combines the selected complementary crisp classifiers using Boolean functions. The first combination approach called Weighted Pruning Pair-wise Boolean Combination (WPBC2), shown in Algorithm 2, combines all possible pairs of complementary crisp classifiers (selected from Phase1 and Phase2) same as in BBC2. In contrast with BBC2, WPBC2 fuses only the complementary crisp classifiers instead of using Brute-force (i.e., all available crisp classifiers). The second approach called Weighted Pruning Iterative Boolean Combination (WPIBC), shown in Algorithm 3, combines the complementary crisp classifiers of each diverse base soft classifiers sequentially same as in IBC. The difference is that WPIBC only combines the most diverse base soft classifiers after pruning all the redundant soft classifiers. As we will show in the evaluation section, both WPBC2 and WPIBC Boolean combination approaches using only $M \ll N$ complementary crisp classifiers of

$L \ll K$ diverse base soft classifiers improved the true positive rate when the false tolerance is almost close to zero.

Algorithm 2: $WPBC2(S_1, \dots S_K, T_1, \dots T_K, lab)$: Weighted Pruning Pair-wise Boolean Combination

input: scores of K soft classifiers $\{S_1, \dots S_K\}$ on a validation set along with their thresholds $\{T_1, \dots T_K\}$, and true labels lab of size $|lab|$.

output: a new composite $ROCCH$—consturcted by $|P_e|$ (size of P_e) combination responses or $|P_e|$ emerging points. Each point is a combination of two crisp classifiers using only one Boolean function.

prune redundant soft and crisp classifiers
$(B < S_1, \dots S_L >, \theta < \theta_1, \dots, \theta_L >) \leftarrow PSCDs(S_1, \dots S_K, T_1, \dots T_K, lab)$
// where $L \ll K$ is the number of selected diverse base soft classifiers
set *BooleanFunctions* $\leftarrow \{a \wedge b, \neg a \wedge b, a \wedge \neg b, \neg(a \wedge b), a \vee b, \neg a \vee b, a \vee \neg b, \neg(a \vee b), a \oplus b, a \equiv b\}$
let $F \leftarrow$ number of Boolean functions in *BooleanFunctions*
let $m_i \leftarrow$ number of decision thresholds in θ_i
let $M \leftarrow \sum_{i=1}^{L} m_i$ total number of crisp classifiers
allocate an array $C[|lab|, M]$
// convert soft classifiers to crisp classifiers
for $i \leftarrow 1$ **to** L **do**
    for $j \leftarrow 1$ **to** m_i **do**
        $r \leftarrow S_i \geq t_j$ **//temporary responses at decision threshold $t_j \in \theta_i$**
        push r onto C
allocate an array $P[2, C^2 \times F]$
// temporary store points (*fpr, tpr*) of fused responses
foreach $bf \in$ *BooleanFunctions* **do**
    for $i \leftarrow 1$ **to** M **do**
        for $j \leftarrow 1$ **to** M **do**
            $r \leftarrow bf(C[i], C[j])$ **// combine responses**
            compute $p \leftarrow$ (*tpr*, *fpr*) using r and *lab*
            push p onto P
compute composite $ROCCH$ of all *ROC* points in P
map each emerging points P_e on $ROCCH$ into a 3-tuples:
$P_e \leftarrow < (S_i, t_j), (S_i, t_j), \text{bf} >$ **//where $i = \{1, \dots, L\}$ and $t_j \in \theta_i$**
return *ROCCH* along with all emerging points $\{P_1, \dots, P_e\}$

Algorithm 3: $WPIBC(S_1, \dots S_K, T_1, \dots T_K, lab)$: Weighted Pruning Iterative Boolean Combination

input: scores of K soft classifiers $\{S_1, \dots S_K\}$ on a validation set along with their thresholds $\{T_1, \dots T_K\}$, and true labels lab of size $|lab|$.

output: a new composite $ROCCH$—consturcted by $|R_{iter}|$ (size of R_{iter}) combination responses or $|R_{iter}|$ emerging points. Each point is a sequential combination on average of five crisp classifiers using four Boolean functions.

call pruning function **//prune redundant soft and crisp classifiers**
$(B < S_1, \dots S_L >, \theta < \theta_1, \dots, \theta_L >) \leftarrow PSCDs(S_1, \dots S_K, T_1, \dots T_K, lab)$
// where $L \ll K$ is the number of selected diverse base soft classifiers
set *BooleanFunctions* $\leftarrow \{a \wedge b, \neg a \wedge b, a \wedge \neg b, \neg(a \wedge b), a \vee b, \neg a \vee b, a \vee \neg b, \neg(a \vee b), a \oplus b, a \equiv b\}$
iter $\leftarrow 1$
// combine the first two ROC curves of the first two diverse base soft classifiers
let $m_1 \leftarrow$ number of points in first curve $ROC(S_1, \theta_1)$
let $m_2 \leftarrow$ number of points in second curve $ROC(S_2, \theta_2)$
allocate an array $P[2, m_1 \times m_2]$ **//temporary store the points of fused responses**
foreach $bf \in$ *BooleanFunctions* **do**
  for $i \leftarrow 1$ **to** m_1 **do**
    $r_1 \leftarrow S_1 \geq t_i$ **// temporary responses at decision threshold $t_i \in \theta_1$**
    for $j \leftarrow 1$ **to** m_2 **do**
      $r_2 \leftarrow S_2 \geq t_j$ **//temporary responses at decision threshold $t_j \in \theta_2$**
      $r_{12} \leftarrow bf(r_1, r_2)$ **// fuse responses**
      compute $p \leftarrow (tpr, fpr)$ using r_{12} and *lab*
      push p onto P
compute $ROCCH_{iter}$ of all combination *ROC* points in P
map each emerging points p_e on $ROCCH_{iter}$ into a 3-tuples:
$p_e \leftarrow < (S_1, t_i), (S_2, t_j), \text{bf} >$
store all emerging points p_e on $ROCCH_{iter}$ onto $R_{1:2}$
// combine rest of the ROC curves of rest of the L-2 diverse base soft classifiers
for $b \leftarrow 3$ **to** L **do**
  let $n_e \leftarrow$ number of emerging points in $R_{1:b-1}$
  let $m_b \leftarrow$ number of points in $l\ ROC_b(S_b, \theta_b)$ curve
  allocate an array $P[2, n_e \times m_b]$ **//temporary storage of fused responses**
  foreach $bf \in$ *BooleanFunctions* **do**
    for $i \leftarrow 1$ **to** n_e **do**
      $r_1 \leftarrow R_{1:b-1}(i)$ **// responses from immediate previous combinations**
      for $j \leftarrow 1$ **to** m_b **do**
        $r_2 \leftarrow S_b \geq t_j$ **//temporary responses at decision threshold $t_j \in \theta_b$**
        $r_{12} \leftarrow bf(r_1, r_2)$ **// fuse responses**
        compute $p \leftarrow (tpr, fpr)$ using r_{12} and *lab*
        push p onto P
  update $ROCCH_{iter}$ of all combination *ROC* points in P
  map each emerging points p_e on $ROCCH_{iter}$ into a 3-tuples:
  $p_e \leftarrow < R_{1:b-1}(i), (S_b, t_j), \text{bf} >$
  store all emerging points p_e on $ROCCH_{iter}$ onto $R_{1:b}$
store all the emerging points to reach on the final $ROCCH_{iter}$ onto $R_{iter} \leftarrow \text{R}_{1:L}$
set *maxiter* and *tol* **// maximum number of iterations and tolerance**
iter $\leftarrow 2$ to *maxiter*
  repeat steps 2 to 33 with $L + 1$ ROC curves: $ROC(R_{iter-1})\ and\ ROC(S_1, \theta_1), \dots, ROC(S_L, \theta_L)$
  if $(AUCH_{iter} \leq AUCH_{iter-1} + tol)$ **then**
    break **// stop further iteration**
return $ROCCH_{iter}$ and R_{iter}

3.2.2 Complexity Analysis

Suppose, we have K soft classifiers with $S_k\{k = 1 \dots K\}$ scores using a validation set V. Let the number of decision thresholds on the scores S_k of each soft classifier is constant and the size is T. And let $N = K * T$ be the total number of crisp classifiers.

The brute-force search for optimal combination is infeasible in practice due to the doubly exponential combinations. In fact, for N crisp classifiers there are 2^N possible outcomes that can be combined in 2^{2^N} ways, which makes the brute-force combination impractical even for small N values [10] [47]. The worst-case time complexities of the proposed and existing Boolean combination methods are given in Table II. The pairwise combination of N crisp classifiers employed in BBC2, which requires $\mathcal{O}(N^2)$ Boolean operations, may not be feasible in practice for large N values. The sequential combination of the IBC algorithm reduces its worst-case time complexity to $\mathcal{O}(T^2 + N)$ Boolean operations.

Table II: The Worst-Case Time Complexity of Pruning and Without Pruning based Boolean Combination Methods

Methods	Pruning	Boolean Combination
BBC2	NA	$\mathcal{O}(N^2)$
IBC	NA	$\mathcal{O}(T^2 + N)$
PBC	$\mathcal{O}(N(\log N + 1))$	$\mathcal{O}(U^2)$
WPBC2	Phase1: $\mathcal{O}(K^2)$ Phase2: $\mathcal{O}(K * (T(\log T + 1)))$	$\mathcal{O}(M^2)$
WPIBC		$\mathcal{O}(m^2 + M)$

The recent pruning approach [12] used the kappa-error diagrams or simply called unweighted kappa coefficient to decide which ensemble members can be pruned with maintaining a similar overall accuracy. Although PBC reduces the impractical exponential computation time for BBC2 to $\mathcal{O}(N(\log N + 1))$, the performance at low false alarm values is also decreased (details in Section

3.3). This is because PBC selects $U \ll N$ complementary crisp classifiers over the whole N converted crisp classifiers, it cannot consider the diversity among the individual soft classifiers.

The proposed pruning technique is more general as it ensures the diversity among both of the individual soft and crisp classifiers instead of using N crisp classifiers. As shown above, the total number of crisp classifiers, *N*, depends on two important parameters *K* and *T*. Phase1 in the proposed weighted pruning technique reduces the size of the ensemble from *K* to *L* diverse soft classifiers, by pruning the redundant ones. As shown in Figure 3.2 (a), out of *K=20* soft HMM classifiers, Phase1 selects only *L=8* HMMs for ADFA-LD dataset and only *L=3* HMMs for CANALI-LD dataset (Figure 3.2 (b)). Then, Phase2 optimizes the size of *T* of each selected base diverse soft classifier (*L*) to $m << T$ by pruning all the trivial and redundant crisp classifiers. Here, m is a user defined parameter and set based on the experimental results using validation set (e.g., in this experiment, $m = 12$ gives the best result for both datasets). At the end, the proposed pruning methods always selects $M = L * m$ complementary crisp classifiers.

Therefore, the worst-case time complexity required by the proposed pruning technique to select M complementary crisp classifiers is $\mathcal{O}(K^2 + K * (T(\log T + 1)))$; where, Phase1 requires about K^2 operations for computing and sorting the AUC and the weighted kappa of *K* soft classifiers, in order to select *L* diverse base soft classifiers. And in Phase2, each base diverse soft classifier (*L*) requires about $T(\log T + 1)$ operations for computing and sorting the unweighted kappa for T crisp classifiers, in order to select $m \ll T$ complementary crisp classifiers. Therefore, in case of worst-case, Phase1 selects all K soft classifiers (i.e., $L = K$). So, the worst-case time complexity for Phase2 requires about $(K * (T(\log T + 1)))$ operations, in order to select a total of $M = K * m$ complementary crisp classifiers. At the end of pruning Phases, Phase3 combines the

decisions of M complementary crisp classifiers. In Phase 3, the worst-case time complexity for the proposed weighted pruning pairwise Boolean combination (WPBC2) is about $\mathcal{O}(M^2)$ Boolean operations and for the proposed weighted pruning iterative Boolean combination (WPIBC) is about $\mathcal{O}(m^2 + M)$ Boolean operations, where $M \ll N$ and $m \ll T$.

3.3 Experiments and Comparison

We experimented with the proposed pruning approach on two system call datasets: ADFA Linux Dataset (ADFA-LD) [7] and CANALI Window Dataset (CANALI-WD) [84]. The experimental results are compared with BBC2 [10] and IBC [11] without pruning. We also compared our approach to PBC that we proposed in previous work [12].

ADFA-LD dataset: ADFA-LD consists of normal and anomalous sequences of system calls collected from Ubuntu [7]. A normal sequence of system calls of a process is collected from the system call traces while it is executed under the normal conditions. An anomalous sequence of system calls of an attack is collected from the system call traces while it is executed against the system. There are in total 5,206 normal traces collected from various normal Unix-based processes such as web browsing and Latex document preparations. The dataset contains 60 attack traces by exercising six different types of attacks: web-based exploitation, simulated social engineering, poisoned executable, remotely triggered vulnerabilities, remote password brute-force attacks, and system manipulation. In training, we use the 833 normal traces same as in [7] to train the 20 discrete-time ergodic HMMs (i.e., K=20 soft classifiers) with various values. The rest of the 4373 normal traces and the 60 anomalous traces are used for evaluation.

CANALI-WD dataset: CANALI-WD consists of two normal datasets called goodware and anubis-good and two malware datasets called malware and malware-test [84]. The goodware

dataset contains a massive amount of 180 GB execution traces of normal day-to-day operations which are collected from 10 different machines. The anubis-good dataset contains the traces of 36 benign applications executed under Anubis [86]. The malware dataset is a collection of execution traces of 6,000 malware samples including a mix of all the existing categories (botnets, worms, dropper, Trojan horses, etc.), which are randomly extracted from Anubis [86]. The final malware-test dataset is a collection of execution traces of 1,200 malware samples which are collected from a different machine than the normal ones used for Anubis. In training, we use the anubis-good dataset and the traces for nine out of 10 machines in the goodware dataset (same as in [84]) to train 20 soft HMMs classifiers with various values. In contrast to [84], however, where the malware dataset was also used to train the models, we only use malware for testing. This is because an anomaly classifier mainly models the normal behavior of a system. Therefore, the rest of the 23 traces of the tenth machine in the goodware dataset, 5,855 traces from malware dataset, and 1,133 traces from malware-test dataset are used for evaluation.

3.3.1 Experimental Setup

We use a stratified 5-Fold Cross Validation (5FCV) technique, same as in [47], on the testing set for the evaluation of the proposed pruning approach. Since the ratio between the normal and anomalous traces in both datasets is not balanced, we applied stratified 5FCV to partition the normal and anomalous sets separately. This is because we want to keep the same ratio (normal to anomalous) to guarantee that all folds include the normal and anomalies traces. Therefore, for ADFA-LD dataset, each fold contains 874 traces selected randomly from the 4373 normal traces and 12 attacks traces selected randomly from the 60 attack traces. Similarly, for CANALI-WD dataset, each fold contains four traces selected randomly from the 23 normal traces and 1,397 traces selected randomly from the 6,988 anomalous traces. However, as we followed the same

setting as in PBC [12] instead the way of standard cross validation, we also used one fold for validation and the remaining four folds for testing on the both ADFA-LD and CANALI-WD datasets.

As described at Section 2.6 in Chapter 2, we apply the BW algorithm on the validation set to learn the parameters of an HMM with setting the random initialization of A, B and π, and M = 340 distinct system call symbols for ADFA-LD dataset and M = 89 distinct symbols for CANALI-WD dataset. Since a single HMM with a predefined number of states N may have limited chances to fit the underlying structure of the data (as noted in Section III), 20 different discrete-time ergodic HMMs (i.e., 20 soft classifiers) are trained with various $N = 10, 20 \ldots 200$ values. For each state value N, we repeated the training process ten times with a different random initialization of A, B and π to avoid the local minima, and the HMM that gives the highest AUC value on the validation set is selected for Boolean combination.

3.3.2 Results and Comparisons

We mainly focus on how the proposed pruning based Boolean combination approaches can reduce the computation time (as discussed in Section V) of the BBC2 and IBC techniques while maintaining or improving the detection accuracy and reducing the false alarm rate.

Figure 3.5 and Figure 3.6 show the AUC results in the ROC space for the proposed weighted pruning techniques on ADFA-LD and CANALI-WD datasets. We can see that the ROC curve of the proposed pruning based WPIBC shows slightly better AUC than IBC. In particularly, WPIBC is able to ensure the diversity among the fused crisp classifiers (selected using unweighted kappa at Phase 2 in Algorithm 1) where each crisp classifier comes from the selected diverse base soft classifiers (selected using weighted kappa at Phase 1 in Algorithm 1). Therefore, in contrast to

IBC, where the order of combination responses in each iteration is the order of all the available soft and crisps classifiers, WPIBC maintains the order of combination responses in each iteration among the selected diverse soft and crisp classifiers (see details in Algorithm 1 and Algorithm 3). For instance, to achieve the final operating points denoted in Figure 3.5 with a large pink circle, WPIBC uses only five selected complementary crisp classifiers (red bold plus marker points) and four Boolean operations, whereas IBC uses 17 crisp classifiers (black bold circle marker points) and 16 Boolean operations.

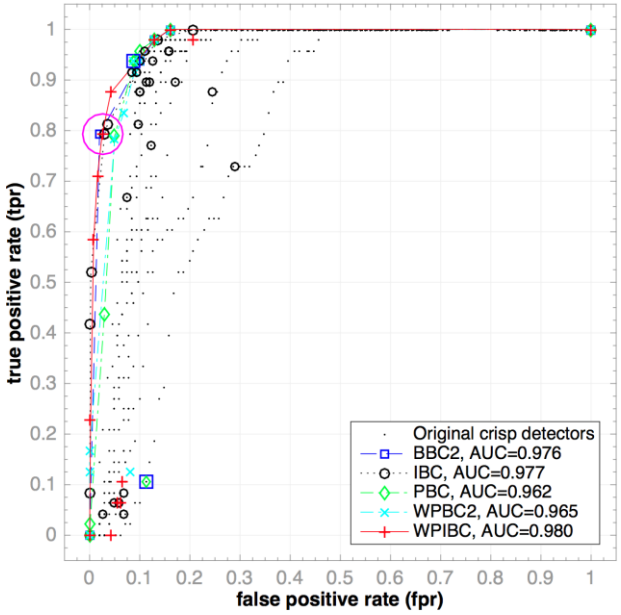


Figure 3.5. Algorithm comparisons on ADFA-LD dataset where one-fold is used for validation and four folds are used for testing.

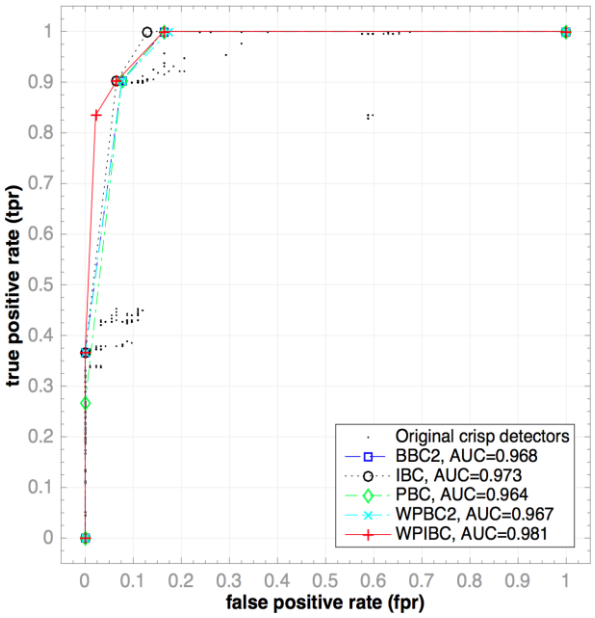


Figure 3.6. Algorithm comparisons on CANALI-WD dataset where one-fold is used for validation and four folds are used for testing.

Compared to BBC2, although the AUC of WPBC2 is slightly low, WPBC2 maintains the same AUC of PBC shown in Figure 3.5 and Figure 3.6. However, WPBC2 overcomes the exponential time complexity problem of BBC2 by pruning the redundant and trivial crisp classifiers, in fact, without pruning, BBC2's time complexity is exponential with respect to the number of classifiers (N^2) [10] [[47].

Table II shows the maximum detection accuracy (*tpr*) achieved by each technique for a fixed (almost close to zero) *fpr* value of 0.002, all values are averaged over the 5FCV.

For ADFA-LD dataset, although the AUC values of all pruning methods are almost equal, the *tpr* of PBC with MinMax-Kappa pruning technique is the worst. The *tpr* of WPIBC is almost equal to that of BBC2 method, and slightly better than that of IBC method. Moreover, the standard deviation of WPIBC is also good as compared to the other methods. For the CANALI-WD dataset, the *tpr* of WPIBC is still better than PBC and WPBC2 pruning techniques, and almost equal to BBC2 and IBC that do not use pruning techniques. Through this analysis, we observed that

Table III: Average (avg), maximum (max), and minimum (min) AUC values and true positive rate (tpr) with false positive rate (fpr)<=0.002, and their standard deviations (std) over the 5FCV (train on one-fold and tested on four folds).

		AUC values				*tpr* with *fpr*<=0.002			
Datasets	**methods**	**avg**	**max**	**min**	**std**	**avg**	**max**	**min**	**std**
	without pruning methods								
ADFA-LD	BBC2	0.98006	0.9852	0.9731	0.0044	0.38334	0.5	0.2292	0.1246
	IBC	0.979	0.983	0.972	0.0042	0.25414	0.4792	0.1665	0.1329
CANALI-WD	BBC2	0.96824	0.9726	0.9601	0.0049	0.36716	0.3739	0.3618	0.0046
	IBC	0.97156	0.9799	0.9612	0.0069	0.36716	0.3739	0.3618	0.0046
	with pruning methods								
ADFA-LD	PBC	0.96762	0.9766	0.9608	0.0078	0.09576	0.2297	0.0208	0.0877
	WPBC2	0.96604	0.9741	0.9602	0.0059	0.11886	0.246	0.054	0.0785
	WPIBC	**0.97762**	0.9788	0.9767	**0.0007**	**0.37498**	0.5208	0.2083	**0.0474**
CANALI-WD	PBC	0.96808	0.9726	0.9601	0.0049	0.24197	0.2639	0.2118	0.0046
	WPBC2	0.96816	0.9729	0.9601	0.0051	0.27716	0.3739	0.2218	0.0071
	WPIBC	**0.96994**	0.9808	0.9541	**0.0021**	**0.34462**	0.3739	0.3225	**0.0034**

Table IV: Average (avg), maximum (max), and minimum (min) AUC values and true positive rate (tpr) with false positive rate (fpr)<=0.002, and their standard deviations (std) over the 5FCV (train on four folds and tested on one-fold).

		AUC values				*tpr* with *fpr*<=0.002			
Datasets	**methods**	**avg**	**max**	**min**	**std**	**avg**	**max**	**min**	**std**
	without pruning methods								
ADFA-LD	BBC2	0.98918	0.99945	0.9829	0.0043	0.4500	0.5833	0.2500	0.1263
	IBC	0.99112	0.9939	0.9887	0.0021	0.41668	0.5000	0.3333	0.0589
CANALI-WD	BBC2	0.97288	0.9963	0.9469	0.0208	0.58648	0.9142	0.3591	0.2963
	IBC	0.98274	0.9981	0.9679	0.0127	0.60722	0.9142	0.3694	0.2798
	with pruning methods								
ADFA-LD	PBC	0.95648	0.9626	0.9533	0.0037	0.0000	0.0000	0.0000	0.0000
	WPBC2	0.9661	0.9703	0.9643	0.0024	0.16666	0.3333	0.0000	0.1317
	WPIBC	**0.98724**	0.992	0.9827	0.0033	**0.49998**	0.5833	0.3333	0.1020
CANALI-WD	PBC	0.97288	0.9963	0.9469	0.0208	0.58648	0.9142	0.3591	0.2963
	WPBC2	0.9736	0.998	0.9469	0.0217	0.58648	0.9142	0.3591	0.2963
	WPIBC	**0.98028**	0.9981	0.9647	0.0151	**0.5981**	0.9142	0.3591	0.2034

the proposed weighted pruning technique combines the selected complementary crisp classifiers iteratively (i.e., called WPIBC), it achieves similar results to that of IBC. Particularly, when we compared the results with the *tpr* where the maximum *fpr* is almost equal to zero (0.002), both WPIBC and WPBC2 outperform PBC. And the results demonstrate that the proposed pruning approach is more general and applicable to either pair-wise Boolean combinations (WPBC2) and iterative Boolean combinations (WPIBC).

Moreover, we tested the proposed pruning approach by using the standard way of 5FCV that is four folds are used in validation and one-fold is used in testing. With this setting, the results of one-fold of 5FCV are demonstrated in Figure 3.7 for ADFA-LD dataset and in Figure 3.8 for CANALI-WD dataset. Table IV shows the average results over the 5FCV with this standard setting

of 5FCV.

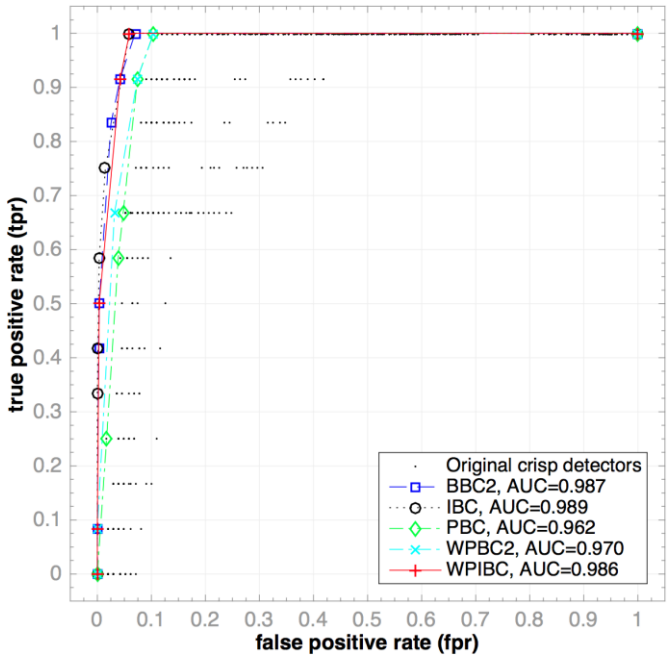


Figure 3.7. Algorithm comparisons on ADFA-LD dataset where four folds are used for validation and one-fold is used for testing in 5FCV.

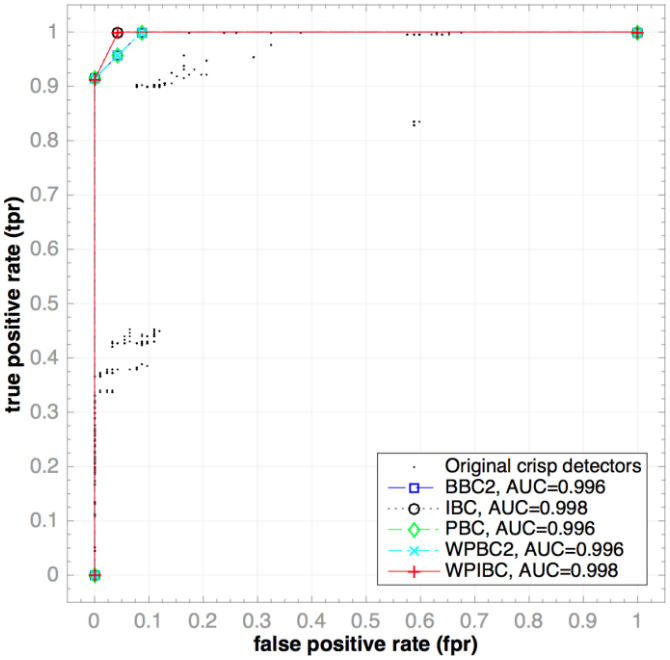


Figure 3.8. Algorithm comparisons on CANALI-WD dataset where four folds are used for validation and one-fold is used for testing.

From Figure 3.7 and Figure 3.8, we can see that for both datasets our proposed pruning based Boolean combination approaches is able to achieve the same performance (in terms of AUC, *fpr* and *tpr*), while reducing the time complexity, the number of crisp classifiers, and the number of Boolean combinations. For CANALI-WD dataset, we got almost equal results with the original approaches (which use all crisp classifiers), and the highest value of *tpr* = 0.91 when the false alarm rate is zero, given in Table IV. However, for ADFA-LD dataset, we observed a great difference between the proposed pruning approach and the PBC. When the average *tpr* = 0.49 for WPIBC (with the limit of maximum *fpr* is equal to 0.002), it is equal to zero for PBC and 0.17 for WPBC2. For example, from the Figure 3.7, we got *tpr* =0.51 (when *fpr*<=0.002) for WPIBC, it is still zero for PBC.

3.3.3 Cost Aalysis

Table V shows the cost that is the combination time and the number of Boolean operations is required by each method during the validation and testing phases. The values are averaged over

Table V: Cost Analysis (Values are Averaged Over 5FCV) in Terms of Pruning and Combination Time (s), and Number of Boolean Operations Applied during Validation Phase, and the Number of Combined Crisp Classifiers Required to Achieve each Vertex on ROCCH during Testing Phase

Methods	Validation phase			Testing phase
	Pruning time (s)	Combination time (s)	# Boolean operations	# combined crisp classifiers
BBC2	NA	16364	4,000,000	2
IBC	NA	11	11,000	11
PBC	1.6	15	19, 701	2
WPBC2	1.9	19	21,701	2
WPIBC	1.9	6	5, 000	5

the 5FCV on the ADFA-LD dataset. All 5FCV executions are performed on a 3.1 GHz Intel Core i7 CPU machine with 16 GB of RAM and a 17x5400 rpm hard disk.

We can see that although the pruning time of the proposed approach is slightly more than the PBC, WPIBC reduced the combination time and the number of Boolean operations to almost half compared to IBC. The total computation time, including pruning and combination during validation of WPIBC was 7.9 seconds whereas PBC took 16.6 seconds. Furthermore, in testing, WPIBC also reduced the number of combined crisp classifiers by almost half than the number required by IBC (5 instead of 11). We can see in Figure 3.5 that WPIBC requires on average five crisp classifiers while IBC requires 11 crisp classifiers to achieve a single point on the final composite ROCCH. Similarly, WPBC2 always requires only two crisp classifiers similar to BBC2 and PBC to achieve a single point on the final ROCCH. Therefore, the proposed pruning approach is more general, and it can be applicable to both pair-wise and iterative Boolean combinations. However, based on the computation time and the number of combined Boolean operations, WPIBC is more desirable in order to obtain better accuracy while reducing the false alarm rates (as shown in Table III and Table IV).

From the worst-case time complexity given in Table II, we can see that the proposed pruning approach reduces the total number of crisp classifiers i.e., $N = K * T$ by optimizing two important parameters of K and T in Phase1 and Phase2 respectively. For example, Phase1 of the proposed weighted pruning approach selects only $L = 3$ diverse ensembles of HMM soft classifiers out of $K = 20$ HMM soft classifiers (shown in Figure 3.2 (b)) for CANALI-WD dataset. As a result, Phase2 computes the unweighted kappa only for about 300 (i.e., $L * T$ and let say $T = 100$) crisp classifiers, in order to select only $M = 36$ (i.e., $M = L * m$, where $m = 12$) complementary crisp classifiers. Whereas, PBC always computes the unweighted kappa for about $N = 2000$ (i.e., $N = K * T$) crisp classifiers, in order to select $U = 50$ complementary crisp classifiers. Moreover, since PBC cannot ensure the diversity among the ensembles of soft HMM classifiers, the probability of

selecting the redundant complementary crisp classifiers or rejecting the other diverse crisp classifiers is also high. In fact, it is reported in Table III and Table IV that PBC significantly reduced the *tpr* with a low false alarm as compared to the other approaches due to the rejection of some diverse complementary crisp classifiers.

3.4 Effects of Weighted Pruning Based Boolean Combination

For any ensemble based Boolean combination algorithms, increasing the accuracy is highly dependent on the diversity among the fused soft/crisp classifiers (i.e., the level of disagreement among the fused soft/crisp classifiers should be high). Although the existing ensemble based BBC2 and IBC Boolean combination techniques implicitly fused such diverse soft/crisp classifiers and showed higher accuracy, they face the challenges of computation time and complexity because of fusing all the possible pair of crisp classifiers from all the available soft classifiers (as discussed in Section 3.2; and reported in Table III). In addition, the accuracy of IBC is also dependent on the order of combinations. In fact, with the increase of number of soft classifiers, the computation time and complexity increase exponentially for BBC2 and linearly for IBC (discussed in Section 3.4).

To be clear, we tested the proposed approach using 50 available soft HMM classifiers (i.e., on average 5000 crisp classifiers), trained with various $N = 5, 10, \ldots 250$ values on CANALI-WD dataset. The results are shown in Figure 3.9, where the values are transformed into a logarithmic scale. It is clear that when we apply the proposed weighted pruning approach (top one in Figure 3.9), a noticeable improvement can be observed in the reduction of the number of Boolean operations. Particularly, WIBC significantly reduces the number of Boolean operations as compared to other approaches. For example, BBC2 requires 25 million (7.4 in logarithmic scale) Boolean operations for 50 soft classifiers, whereas, WPBC2 uses only 3,600 (3.4) operations.

Similarly, when IBC requires 15 thousand (4.2 in logarithmic scale) Boolean operations, WIBC uses only 204 (2.3) operations. As a result, we can state that WPBC2 6944 times faster than BBC2 and WPIBC 73 times faster than IBC for 50 soft classifiers. Moreover, from the 30 soft classifiers, WPBC2 and WIBC always select five diverse soft classifiers with the increase of the number of soft classifiers, and thus, reach a constant number of Boolean operations.

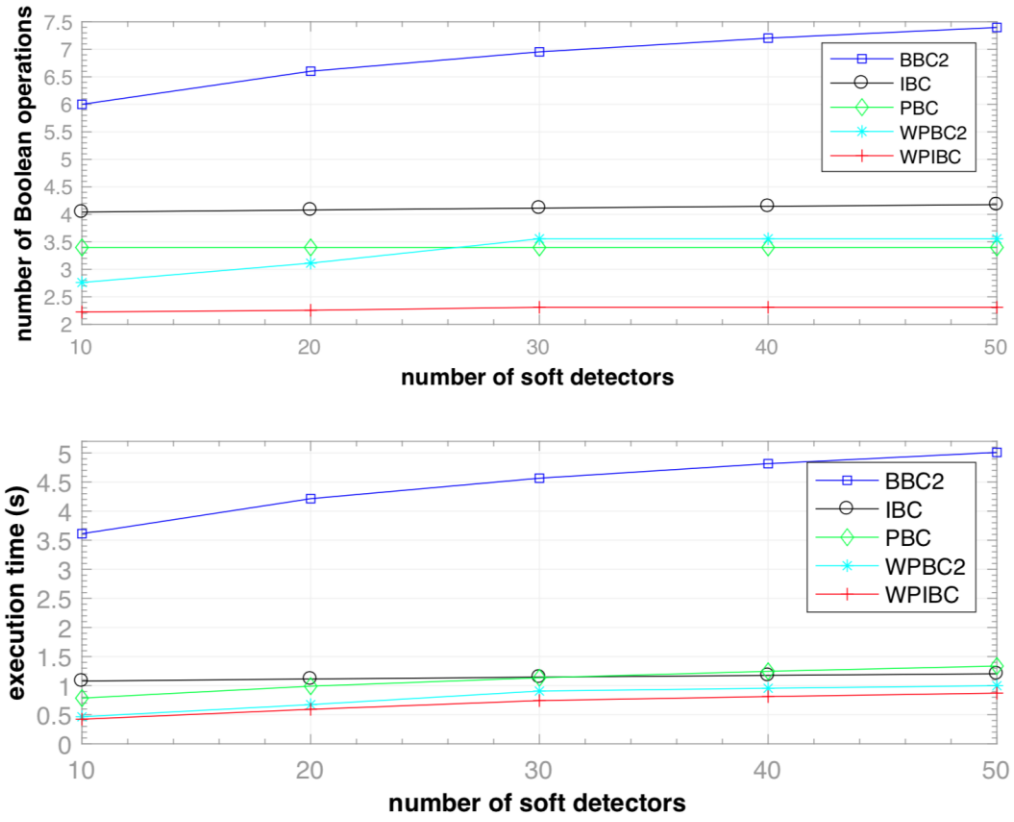


Figure 3.9. Algorithm's computation time and complexity analysis on the validation subset of CANALI-WD dataset

The bottom part of Figure 3.9 compares the computation time (including pruning and combination time together for pruning based approaches). We can see that WPBC2 and WIBC reported the lowest computation times as compared to other approaches. For example, WPBC2 is ten thousand times (seconds) faster than BBC2 and WPIBC is two times faster than IBC during

validation phase using 50 available soft classifiers. Moreover, from the 30 soft classifiers, although the pruning time for WPBC2 and WIBC increase slightly with the increase of number of soft classifiers, the combination time remains same as both are always using only five selected diverse soft classifiers. Compared to PBC pruning approach where the pruning and combination time both are increasing linearly with the increase of number of soft classifiers.

In fact, PBC shows worst result when the false alarm is almost zero for both ADFA-LD and CANALI-WD datasets (given in Table III and Table IV). On the other hand, the accuracy with almost zero false alarm is the desired expected solution for deploying an ADS in a real-world application. The reason is that PBC also uses all the available soft classifiers to select a subset of complementary crisp classifiers without ensuring the diversities among the use of soft classifiers. As a result, the redundant soft classifiers produce redundant crisp classifiers, and thus it increases the probability of selecting these redundant copies if anyone is selected as a complementary crisp classifier by MinMax-Kappa pruning technique of PBC.

The proposed WPBC2 and WPIBC weighted pruning techniques select the most diverse base soft classifiers from the available soft classifiers using weighted kappa. For instance, from the Figure 3.3 (a), eight diverse base soft classifiers are selected while 12 are pruned as for redundant copies for ADFA-LD dataset. Similarly, from the Figure 3.3 (b), only three diverse base soft classifiers are selected while 17 are pruned for CANALI-WD dataset. As the selected base soft classifiers are diverse, the converted crisp classifiers from them might also be diverse as well. Therefore, when we apply the MinMax-Kappa pruning technique on each selected diverse base soft classifiers individually, there has no chance for the selection of redundant complementary crisp classifiers. As a result, our proposed pruning technique shows better accuracy when the false alarm is close to zero compared to PBC for both datasets (given in Table III and Table IV).

Moreover, the proposed weighted based pruning approach is more general as we can combine the selected diverse soft/crisp classifiers either pair-wise or iteratively same as in BBC2 or IBC Boolean combination techniques.

Although the proposed approach is experimentally validated only on HIDS using system call data, it can be applied in other application domains particularly, where one model does not formulate the complex normal behaviors of a system. In that case, we can train ensemble classifiers with considering various normal behaviors. Then, the proposed method may be a good one for pruning and combining the multiple classifier's decisions. For example, detecting programming errors (i.e., software bugs) and root causes in a complex computer programming system [87] [88]. Fosdick et al. [89] reported that a computer program is strongly related to the computation patterns of input data and thus useful for detecting the data flow anomalies. The sequences of operations i.e., the flows of data are assumed to be consistent and used them to model ensembles classifiers. A social or cultural event or road accident can also be detected using sensor and user (e.g., users of twitter, Facebook, etc.) generated data. For example, Pramod et al. [90] trained several linear Markov models by segmenting the non-linear traffic data and used them to detect the city events.

3.5 Limitations and Discussions

Our approach is limited to ensemble of homogeneous soft anomaly classifiers (i.e., multiple HMMs). However, the input can be ensemble of heterogeneous soft and crisp anomaly classifiers (e.g., STIDE [33], SVM [80], etc.). In fact, having different types of classifiers should further increase the diversity in the ensemble and allow for improved performance [23]. Heterogenous classifiers use different learning techniques and may commit different (and potentially complementary) type of errors, which increases the diversity in the ensemble. For example, OC-

SVM models the normal behavior of a system using fixed-size feature vectors instead of sequential features like HMM; STIDE uses the Hamming distance, whereas HMM uses likelihood probability as a matching measure.

To adapt our approach to support heterogeneous classifiers, we need to modify Phase1, which assumes the same thresholds of a base soft classifier, which are the orders or levels for the weighted kappa for computing the diversity score. It may be more efficient to group them based on each modeling technique. Then, apply the Phase1 pruning technique on each group separately. For example, STIDE with various sliding window sizes can be used to produce many homogeneous soft classifiers [33], which can then feed as input to Phase1.

Although the proposed approach significantly reduces the Boolean combination time (see **Error! Reference source not found.**) by pruning the number of combined soft (K) and crisp c lassifiers (N), the worst-case time complexity, particularly, for the pruning phases (given in Table II), will be increased exponentially (K^2) with the increase of K. Therefore, for large values of K, the pruning approach may suffer from scalability problems. To address this limitation, we need resort to parallel processing techniques and platforms such as the Hadoop ecosystem [72] [74].

Moreover, the proposed approach is dependent on the ROC space for pruning and combining the decisions of the selected complementary crisp classifiers. However, here, the used ROC curves is a binary classification problem. Therefore, to extend the approach for multiclass classification problems, we need to work with a ROC curve for more than two classes and then adapt the Boolean combination and pruning techniques to accommodate multiple classes.

3.6 Conclusion

The proposed effective pruning-based Boolean combination techniques analyze the diversities among the available ensemble soft classifiers (HMMs) using weighted kappa (measures the agreement/disagreement between two soft classifiers). Based on the weighted kappa coefficients, it selects a best subset of diverse base soft classifiers while pruned all the redundant soft classifiers. Each selected base soft classifier is then converted into all the possible crisp classifiers (at various decision thresholds) and used them for selecting a subset of complementary crisp classifiers using unweighted kappa-based MinMax-Kappa pruning technique. At the end, we merge all the selected complementary crisp classifiers and use them for Boolean combinations. The experimental evaluation on the two benchmarking ADFA-LD and CANALI-WD system call datasets verified the validation of the proposed method. We achieved much better results than the recent PBC pruning technique, particularly, when the false alarm is almost close to zero.

Our future plan is to investigate the proposed pruning approach using different diverse classifiers and other datasets. Moreover, we also want to leverage Big Data platforms such as Hadoop and the MapReduce programming model in order to further improve the performance of our approach, especially when used with multiple heterogeneous ensemble soft classifiers such as HMMs, One-class SVM, STIDE, and so on.

Chapter 4. EnHMM: On the Use of Ensemble HMMs and Stack Traces to Predict the Reassignment of Bug Report Fields

Bug reports (BR) contain vital information that can help triaging teams prioritize and assign bugs to developers who will provide the fixes. However, studies have shown that BR fields often contain incorrect information that need to be reassigned, which delays the bug fixing process. There exist approaches for predicting whether a BR field will most likely be reassigned or not. These studies use mainly BR descriptions and traditional machine learning algorithms (e.g., SVM, KNN, etc.). As such, they do not fully benefit from the sequential order of information in BR data, such as function call sequences in BR stack traces, which may be valuable for improving the prediction accuracy. In this paper, we propose a novel approach, called EnHMM, for predicting the reassignment of BR fields using ensemble Hidden Markov Models (HMMs), trained on stack traces. EnHMM leverages the natural ability of HMMs to represent sequential data to model the temporal order of function calls in BR stack traces.

We applied EnHMM to BRs from the Eclipse and Gnome systems. For Eclipse, our approach provides an average precision, recall, and F-measure of 54%, 76%, and 60%, respectively. For Gnome, we obtained about 41% precision, 69% recall, and 51% F-measure. We also found that EnHMM improves over the best single HMM by 36% for Eclipse and 76% for Gnome. Furthermore, a comparative study reveals that EnHMM outperforms state-of-the-art techniques including Im-ML.KNN [30], Naïve Bayes [1], ML.KNN [18], and HOMER [8]. These results demonstrate that EnHMM, trained on BR stack traces, holds real promise for predicting BR field reassignments.

4.1 EnHMM Approach

Our approach for predicting the reassignment of BR fields consists of four phases as shown in Figure 4.1: (1) preprocessing, (2) training, (3) validation, and (4) testing. In the preprocessing phase, we extract and profile sequences of function calls from stack traces of BRs. Note that not all BRs come with stack traces, so we only include BRs with stack traces in our dataset. In the training phase, we use temporal sequences of function calls extracted from stack traces to train multiple HMMs for each BR field of interest (e.g., product, component, etc.). In the third phase, the validation phase, we select the most diverse classifiers out of the available HMMs. For this, we use WPIBC [15], which ensures diversity among the combination of multiple classifiers. The selected diverse classifiers are used to construct the proposed ensemble HMMs. In the last phase, the testing phase, we use the constructed Boolean combination rules on each BR field of the testing set of BRs to predict whether it gets reassigned or not.

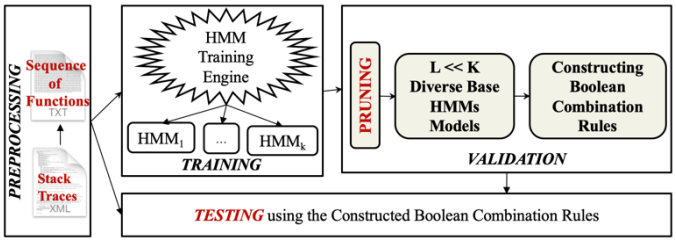


Figure 4.1. An overview of our approach

4.1.1 Extracting and Profiling Sequences of Function Calls from Stack Traces

A stack trace contains a sequence of function calls that are in memory when the crash occurs. In both Eclipse and Gnome bug tracking systems (used in this study), a BR submitter manually appends stack traces to BR descriptions and comments. To extract stack traces, we need to use regular expressions.

Bettenburg et al. [91] implemented a tool (Infozila) to extract stack traces from Eclipse BR descriptions and showed that their regular expression can extract stack traces with 98% accuracy. Lerch et al [67] improved the regular expression proposed by Bettenburg et al. [91] to detect stack traces with a higher accuracy and proposed the following regular expression, which we use in our study:

[EXCEPTION] ([:][MESSAGE])? ([at][METHOD][(] [SOURCE] [)])+ ([Caused by:] [TEMPLATE])?

Similarly, we need to define a regular expression to extract stack traces from BR descriptions in the Gnome bug tracking system. We designed the following regular expression after examining manually hundreds of Gnome BRs:

([#NUMBER] [HEX ADDRESS] [IN] [FUNCTION NAME] [(] [PARAMETERS] [)] ([FROM] | [AT]) ([LIBRARYNAME] | [FILENAME]))*

For each BR, we extract the sequence of function calls in its associated stack traces, which we will use to train multiple HMMs.

4.1.2 Training an HMM

Our approach is used to predict the reassignment of any BR field of interest (e.g., component, product, severity, OS, version, etc.) that we refer to as BR field, F_i.

For a given F_i, we create an HMM by specifying the number of hidden states. The training phase consists of the following steps. We split the BRs into two sets: the BRs that have their field F_i reassigned (R) and those that have their field F_i not reassigned (NR). We use 70% of BRs from R to train the HMM. We use 10% of BRs from R and another 10% of BRs from NR to create the

validation set. For testing (see the next subsection), we use 20% of BRs from R and the remaining 90% of BRs from NR. This way of splitting the data is a common practice in machine learning. This said, a different splitting may yield different results, which constitutes an internal threat to validity of our approach.

The output of this phase is an HMM that learns the pattern of BR-associated stack trace for which field F_i is reassigned. We call this model HMM-R_{Fi}. This model can help predict for a new incoming BR whether field F_i would get reassigned or not. However, the limited number of trained reassigned BRs (i.e., observations from the rare class) on a specific field F_i causes a data imbalance problem as shown by Xia et al. **[56]**. Simply learning a model from the BRs for which Field F_i is reassigned will most likely increase the false positive rate. To address this, we need to create another model that is trained on the major class observations (meaning BRs for which F_i is not reassigned). We create another model, called HMM-NR_{Fi} to represent BRs in the historical data for which F_i is not reassigned. The idea is to combine multiple instances of each model by varying the number of hidden states (see next subsection) into a powerful classifier that knows about both the rare and major class observations. HMM-NR_{Fi} is trained using the same process as HMM-R_{Fi}. We use 70% of NR for training, 10% from R and another 10% from NR for validation. For testing, we use 90% of R BRs and 20% from NR. Figure 4.2 shows how the data is split for training, validation, and testing purposes for both HMM-R_{Fi} and HMM-NR_{Fi} with an example of 10,860 BRs collected from the Eclipse project on 'component' field (given in Table VI).

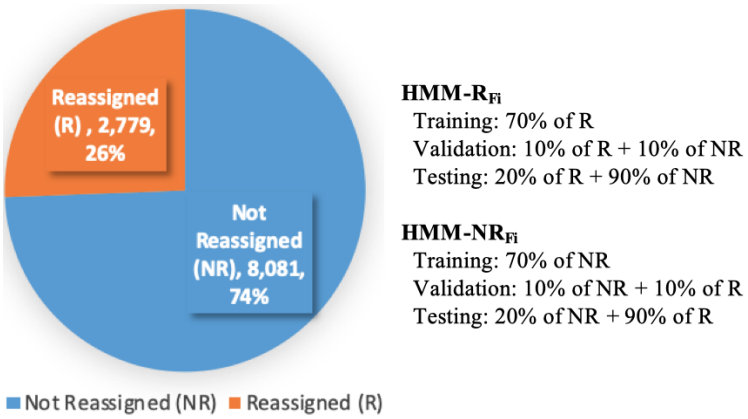


Figure 4.2. Splitting the training, testing, and validation sets from the Eclipse bug reports on field, F_i (i=Component) for HMM-R_{Fi} and HMM-NR_{Fi} models.

4.1.3 Constructing Ensemble HMMs

The proposed ensemble HMMs are composed of HMM-R_{Fi} and HMM-NR_{Fi}; each trained by varying the number of hidden states from N=10, 20…200. As a result, for each field F_i, we will have 20 HMM-R_{Fi} and 20 HMM-NR_{Fi} models combined. To our knowledge, there is no work that precisely defines how many hidden states we should have for best accuracy. Most studies (e.g., [11]) vary the number of hidden states as we propose in this paper.

The combination of these multiple HMM-R_{Fi} and HMM-NR_{Fi} soft classifiers works at the decision label (i.e., '0' for not reassigned and '1' for reassigned). A decision is made by a crisp HMM-R_{Fi} or HMM-NR_{Fi} classifier with a predefined threshold, θ. Assume, in the validation set, we have n BRs for Field F_i. We therefore obtain n scores (S_n) computed by a trained soft HMM-R_{Fi} / HMM-NR_{Fi} classifier. We obtain n responses {R_n: 1 *if* $Sn > \theta$, otherwise 0}, which also represents the number of crisp classifiers. Our HMM decision-level combination technique is based on WPIBC and consists of three steps (as described in Chapter 3): (a) selecting base soft classifiers, (b) selecting complementary crisp classifiers, and (c) constructing Boolean

combination rules.

Selecting Base Soft Classifiers: Suppose, there are k trained HMM-R_{Fi} and HMM-NR_{Fi} soft classifiers and each one produces a set of scores (S_k) of size |V|, where V is the validation set. We use T_k to refer to all possible thresholds on scores. Therefore, we have k ROC curves (S_k, T_k) with k AUC values. Initially, we select a *base* soft classifier $k^* = \max[AUC(k)]$ for which the AUC is the highest. Then we compute agreement coefficients between the base soft classifier (k^*) and all the other soft classifiers. We set an agreement threshold τ to 90% as a default value. This means that soft classifiers that agree 90% with scores computed by the base soft classifier (k^*) are considered redundant, and therefore should be pruned. Assume, we found $k^{\sim}$ redundant copies of the base classifier k^*. So, we select the base k^* and prune $k^{\sim}$ redundant ones. The process is repeated with the remaining $(k - k^{\sim} - k^*)$ soft classifiers and continues until we are left with only one base soft classifier. At the end, we obtain a total of $l << k$ diverse base soft classifiers.

Figure 4.3 shows an example with k=40 trained soft classifiers (i.e., 20 HMM-R_{Fi} and 20 HMM-NR_{Fi}) using the validation set. We can see that only six (i.e., l=6, three from HMM-R_{Fi} and three from HMM-NR_{Fi}) soft classifiers are selected as diverse. All the other ones are pruned because they are redundant. The resulting l=6 base soft classifiers are then used to select the final complementary crisp classifiers.

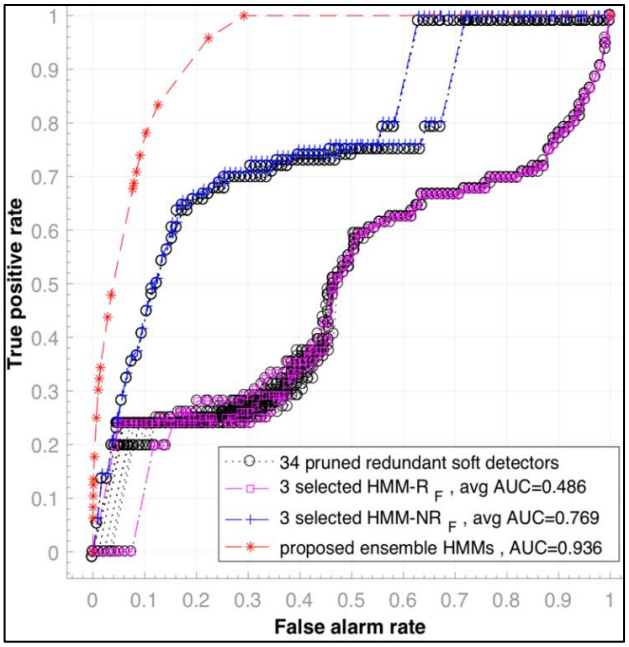


Figure 4.3. Example of selected six diverse base HMM-RFi and HMM-NRFi soft classifiers after pruning all the redundant ones under the ROC space using the validation set.

Selecting Complementary Crisp Classifiers: Suppose we have T_l possible thresholds on scores computed by a base soft $HMM - R_{Fi}^{l}$ or $HMM - NR_{Fi}^{l}$ classifier (l). We therefore obtain T_l candidate crisp $HMM - R_{Fi}^{l}(T_l)$ or $HMM - NR_{Fi}^{l}(T_l)$ classifiers. Then, we compute kappa (kp) agreement coefficients between each crisp classifier's decisions and decisions from the ground truth. The accurate crisp classifiers should be close to $kp \approx kp_{max}$ and their complementary crisp classifiers should be close to $kp \approx kp_{min}$. Assume the number of selected crisp classifiers is D and the ratio between accurate and their complementary crisp classifiers is 50%, we sort candidate crisp classifiers in a descending order based on their kp agreement coefficients. Then, we select the top D/2 (i.e., 50% of total) as accurate crisp classifiers and the bottom D/2 as their complementary ones, respectively.

Constructing Boolean Combination Rules: We combine decisions/responses (0/1)

produced by each selected complementary crisp classifier by leveraging the WPIBC Boolean combination technique [15]. WPIBC uses the same Boolean operators as previous approaches, namely IBC [11], except that it uses only base soft classifiers with their selected complementary crisp classifiers instead of all available candidate soft and crisp classifiers (as it is the case of IBC). We also use ten different Boolean combination functions to combine two crisp classifiers' decisions on the ROC space. Initially, we combine the first two base soft classifiers and then, the resulting emerging responses are combined with the next base one and so on. We repeat this combination process iteratively until no further improvement is reached. The composite ROC curve (red curve in Figure 4.3) with the AUC about 93% is the combination of selected complementary crisp HMM-R_{Fi}/HMM-NR_{Fi} classifiers produced by six selected base soft HMM-R_{Fi}/HMM-NR_{Fi} classifiers using the validation set and θ as a threshold. The constructed Boolean combination rules are then used during testing.

4.2 Case Study Setup and Results

This case study aims to answer the following questions:

- RQ1: How does EnHMM perform in terms of its ability to predict BR field reassignment?
- RQ2: How does EnHMM perform in comparison to a single HMM when predicting BR field reassignment?
- RQ3: How does EnHMM compare to existing techniques?

4.2.1 Datasets

We use Eclipse and Gnome bug repositories to assess the performance of our approach.

Eclipse and Gnome are two open source software systems and their bug repositories are publicly available through Bugzilla bug tracking system. We only consider BRs with status "resolved", "closed" and "fixed". From Eclipse, we collect 83,984 BRs from January 01, 2008 to July 19, 2011, among which 10,860 (12.9%) have stack traces. This exact Eclipse dataset was used by other studies (e.g., Im-ML.KNN [56], ML.KNN [57]). This will help us compare our results with other approaches. For Gnome, we collect 55,438 BRs from December 28, 2007 to July 20, 2011, among which 10,579 (19.08%) have stack traces. This dataset was used by the authors in other studies. (We are currently building larger datasets on which we intend to replicate this work.)

Table VI shows the distribution of reassigned and not reassigned BRs for eight BR fields: Product, Component, Version, OS, Priority, Severity, and Status. As expected, the number of BRs for which field F_i is not reassigned is much higher than the number of BRs that are reassigned, which shows a clear imbalance of the data. As we explained in Section 4.1.2, we address this by creating a model for each class, HMM-R_{Fi} and HMM-NR_{Fi}, and combine them.

Table VI. Statistics on BRs (BR) with Stack Traces Collected from Eclipse and Gnome Bug Repositories

Dataset	Class Label	Assignee		Product		Component		Version		OS		Priority		Severity		Status	
		#BR	*%*	*#BR*	*%*	*#BR*	*%*	*#BR*	*%*	*#BR*	*%*	*#BR*	*%*	*#BR*	*%*	*#BR*	*%*
Eclipse	*Not-Reassigned*	3,566	33	9,156	84	8,081	74	8,875	82	10,194	94	9,702	89	9,593	88	9,451	87
	Reassigned	7,294	67	1,704	16	2,779	26	1,985	18	666	6	1,158	11	1,267	12	1,409	13
Gnome	*Not-Reassigned*	3,752	73	8,813	83	7,930	75	6,612	63	10,471	99	9,404	89	9,317	88	9,736	92
	Reassigned	6,827	27	1,766	17	2,649	25	3,967	37	108	1	1,175	11	1,262	12	843	8

4.2.2 Training HMMs for Field F_i

As discussed in Section 4.1.2, to train an HMM, we split the BRs associated with field (F_i) into two groups: BRs that have F_i reassigned, and those that have F_i not reassigned. Each group is then divided into three sets: training (70%), validation (10%), and testing (20%). The 10%

validation set contains BR traces from each group. For testing, we use 20% of BR traces from the training class and 90% from the other group of BR traces. For example, in Eclipse, the number of stack traces used for training, validation, and testing one HMM-$NR_{Fproduct}$ model, given that the number of BRs with stack traces that have the product field reassigned and not reassigned is 1,704 and 9,156, respectively (see Table VI) is as follows:

- Training set contains 6,409 traces (=9,156*70%)
- Validation set contains 1,086 traces (9,156*10% + 1,704*10%)
- Testing set contains 3,365 traces (=9,156*20% + 1,704*90%)

We apply the same process to HMM-$R_{Fproduct}$ and also to construct HMM-R_{Fi} and HMM-NR_{Fi} for every other field F_i. In addition, for each field F_i, we train 20 HMM-R_{Fi} and HMM-NR_{Fi} by varying the number of hidden states (N), from 10 to 200 with bonds of 10. In total, we built 280 (=40*7) different HMM models for the prediction of the eight BR fields shown in Table VI. Note that not all of these HMM models are used in the actual prediction since the WPIBC (the selected HMM combination approach) prunes the redundant ones.

4.2.3 Evaluation Metrics

In addition to the ROC curve that we discussed in Section III, we also use precision, recall, and F-measure to measure the performance of EnHMM to predict BR field reassignment. These metrics are used in the literature to evaluate the accuracy of a classifier [21] [55] [57] [58].

Precision and recall are defined as follows:

$$Precision = \frac{TP}{TP + FP} \qquad (4.1)$$

$$Recall = \frac{TP}{TP + FN} \quad (4.2)$$

TP: True Positives; FP: False Positives; FN: False Negatives.

Precision is the ratio of the number of BRs that we correctly predicted that their field (Fi) is reassigned (TP) to the total number of BRs for which we predicted that their field (Fi) is reassigned (TP+FP). Recall is the ratio of the number of BRs that we correctly predicted that their field (Fi) is reassigned (TP) to the total number of BRs that actually have their field (Fi) reassigned (TP+FN). To have a better perception of the result, we also use F-measure, a harmonic mean of precision and recall and is defined as follows:

$$F - measure = \frac{2 \times Precision \times Recall}{Precision + Recall} \quad (4.3)$$

4.2.4 Experimental Results

We use the ROC curves (see Figure 4.4 and Figure 4.5) to show the effectiveness of EnHMM in predicting whether a BR field of a new incoming BR would be reassigned or not by addressing RQ1, RQ2, and RQ3.

RQ1. How does EnHMM perform in terms of its ability to predict BR field reassignment?

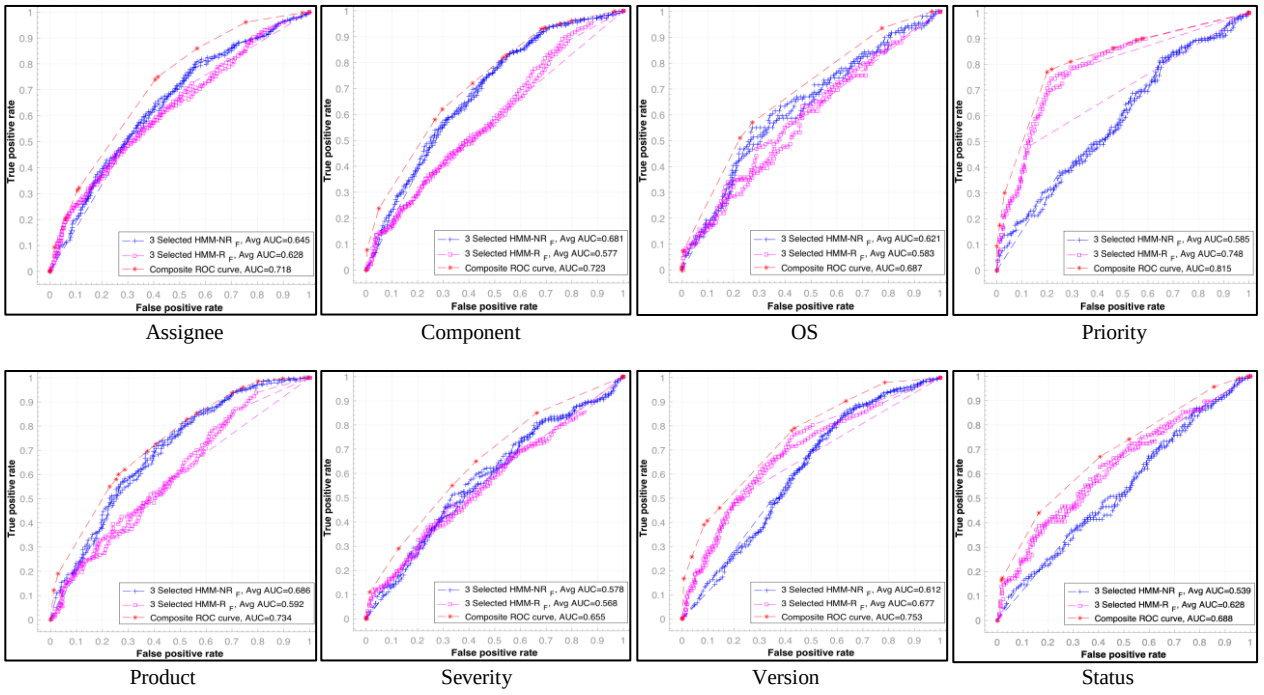


Figure 4.4. Results on the testing set for Eclipse bug report fields

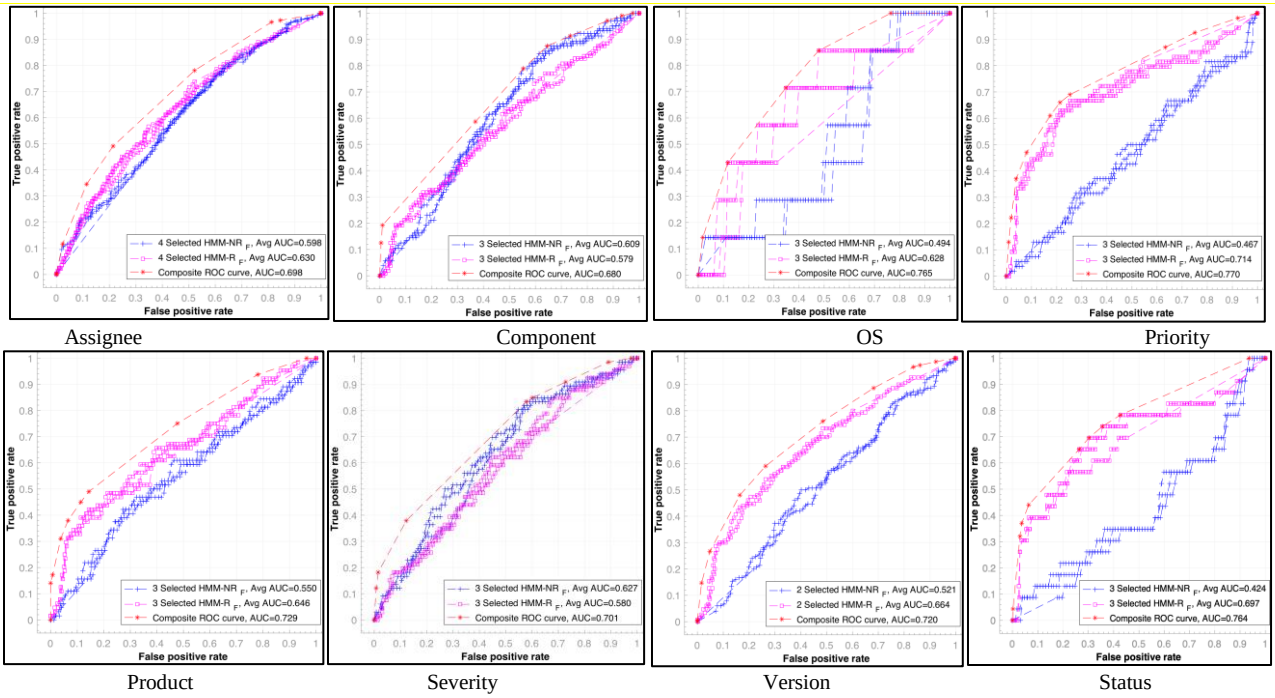


Figure 4.5. Results on the testing set for Gnome bug report fields

We can easily compute the best precision, recall, and F-measure for each predicted BR field F_i from the corresponding ROC curve shown in Figure 4.4 and Figure 4.5. Each point (*fpr*, *tpr*) on the final composite ROC curve produced by EnHMM represents the predicted responses (i.e., the decisions whether the testing BRs will be reassigned (i.e., 1) on field F_i or not reassigned (i.e., 0) on field F_i. We used this set of predicted responses (i.e., a set of points) on the composite ROC curve for Field F_i to compute a set of precisions, recalls, and F-measures using Equations (4.1), (4.2), and (4.3). Finally, a point (i.e., the *tpr* and *fpr* of the responses or predicted outcomes) out of all the points on the ROC curve produced by EnHMM (red one with star marker points) that give the maximum F-measure is selected as the best predictor with a best precision, recall, and F-measure for each BR field F_i.

Table VII shows the best F-measure of the proposed ensemble HMMs for each field F_i from the corresponding ROC curve shown in Figure 4.5 for Eclipse and Gnome datasets. Overall, EnHMM performs relatively well for most cases, with some noticeable exceptions. For example, it only detects the "severity" field with a precision of 21% for Eclipse and 22% for Gnome (the lowest precision obtained). We also notice that for the "status" field, EnHMM achieves a low recall for both Eclipse and Gnome (27% and 35% respectively). This may be due to the low number of BRs for which this field is reassigned as shown in Table VI. On the other hand, we notice a very high

Table VII. Accuracy of EnHMM

BR Field	Datasets	Precision	Recall	F-measure
Assignee	Eclipse	80.15%	97.12%	87.82%
	Gnome	82.69%	95.91%	88.82%
Component	Eclipse	62.50%	67.87%	65.00%
	Gnome	45.61%	100.0%	62.65%
OS	Eclipse	36.82%	100.0%	53.83%
	Gnome	28.71%	100.0%	55.81%
Priority	Eclipse	54.75%	75.63%	63.52%
	Gnome	26.32%	55.56%	35.71%
Product	Eclipse	57.57%	98.90%	72.78%
	Gnome	45.61%	40.63%	42.98%
Severity	Eclipse	21.04%	72.87%	32.66%
	Gnome	22.17%	65.15%	33.08%
Version	Eclipse	61.19%	72.00%	66.16%
	Gnome	50.88%	58.00%	54.21%
Status	Eclipse	57.41%	26.72%	36.47%
	Gnome	28.57%	34.78%	31.37%
Average	**Eclipse**	**53.93%**	**76.39%**	**59.78%**
	Gnome	**41.32%**	**68.76%**	**50.59%**

precision and recall for fields that contain a large number of BRs for which the respective field is reassigned very often. For example, the "assignee" field, which is reassigned in 68% of the BRs for Eclipse and 27% BRs in Gnome can be predicted with 80% precision and 97% recall for Eclipse and 83% precision and 96% recall for Gnome. We need to conduct more studies to understand the reasons behind the performance of EnHMM by examining various factors including the impact of the size of the dataset on the approach, as well as the size and content of the BR stack traces. For now, we state the following finding:

Finding 1:

EnHMM achieves an average precision, recall, and F-measure of 54%, 76%, and 60% on Eclipse dataset and 41%, 69%, and 51% on Gnome dataset.

RQ2. How does EnHMM perform in comparison to a single HMM when predicting BR field reassignment?

From Figure 4.4 and Figure 4.5, we can see that EnHMM (represented with the red curve in the figures) always gives a better accuracy than the best selected single HMM classifier (the blue/pink curves) for all BR fields for both datasets. The ensemble HMMs significantly improves the AUC, while reducing the false positive rates compared to the best single HMM (the ROC curve in blue or pink depending on the field, which is the closest to the EnHMM red curve). For example, for the "assignee" field in Eclipse data (see Figure 4.4), the AUC of the ROC curve corresponding to the three selected HMM-$NR_{assignee}$ is 0.645, the AUC of the three selected HMM-$R_{assignee}$ is 0.628, and the AUC of EnHMM (composite ROC curve) = 0.718. This also shows that the rules constructed by the ten different Boolean combination functions yields good results.

To dig deeper, we analyzed each ROC curve shown in Figure 4.4 and Figure 4.5 on Eclipse and Gnome testing datasets to find the maximum *tpr* at the y-axis against a maximum tolerable *fpr (MTPR)* at the x-axis for each BR field using EnHMM and a single HMM. We measure the improvement as follows:

$$\text{Improvement} = (TPR_{EnHMM} - TPR_{singleHMM}) / TPR_{singleHMM}$$

Table VIII shows the results. For example, for the "assignee" field in Eclipse data, the maximum tolerable FPR (MTFPR) is 12%, the TPR obtained using EnHMM that corresponds to MTFPR in the ROC curve is 32% and that of a single HMM is 26%, which shows that EnHMM results in 32% improvement over the best single HMM.

Table VIII. Improvement of EnHMM over single HMM

BR Field	Datasets	MTFPR	TPR EnHMM	TPR Single HMM	Improvement
Assignee	Eclipse	12%	32%	26%	23%
	Gnome	11%	34%	27%	26%
Component	Eclipse	5%	24%	14%	71%
	Gnome	1%	19%	4%	375%
OS	Eclipse	22%	51%	47%	9%
	Gnome	12%	43%	43%	0%
Priority	Eclipse	2%	30%	22%	36%
	Gnome	8%	47%	42%	12%
Product	Eclipse	2%	19%	12%	58%
	Gnome	14%	49%	42%	17%
Severity	Eclipse	12%	29%	20%	45%
	Gnome	12%	38%	20%	90%
Version	Eclipse	10%	41%	30%	37%
	Gnome	5%	26%	15%	73%
Status	Eclipse	16%	44%	39%	13%
	Gnome	6%	44%	39%	13%
Average	**Eclipse**	**10%**	**34%**	**26%**	**36%**
	Gnome	**9%**	**38%**	**29%**	**76%**

In addition, Figure 4.4 and Figure 4.5 show the number of selected classifiers out of the 40 classifiers (20 HMM-R_{Fi} and 20 HMM-NR_{Fi}) used initially for each field. For example, for the "product", "component", "severity" and "assignee" fields in Eclipse dataset, our approach only needed 6 classifiers (3 HMM-R_{Fi} and 3 HMM-NR_{Fi}) out of 40 to provide optimum AUC (=0.734). The maximum number of selected classifiers (i.e., after the pruning step) independently from any field is 8. We needed a maximum of 5 HMM-R_{Fi} and 3 HMM-NR_{Fi} to attain best accuracy for the prediction of the OS and Priority fields. Similarly, we needed 3 HMM-R_{Fi} and 3 HMM-NR_{Fi} to predict the "component", "OS", "product", "priority" and "severity" fields for the Gnome dataset. In other words, our approach only needed a maximum of 8 out 40 initial classifiers (20%) to predict any of the fields, which suggests that it is not only effective for predicting the reassignment of these fields, but also scalable with the growing number of classifiers.

Finding 2:

EnHMM improves over a single HMM by 36% for Eclipse and 76% for Gnome. In addition, EnHMM requires at most 20% of the initial classifiers thanks to the Kappa-based pruning approach used to prune redundant classifiers.

RQ3: How does EnHMM compare to existing techniques?

We compare our approach with a recent approach proposed by Xia et al. [56], called the imbalanced multi-label k-Nearest Neighbors (Im-ML.KNN). The authors proposed a machine learning approach, which is a composite classifier where each classifier uses the same multi-label KNN (ML.KNN) machine learning algorithm [57] to train the model. The main novelty of Im-ML.KNN is the combination of three classifiers that are built on top of three separate features types: BR field metadata, BR description and summary, and a mix of both. When applied to four large BRs datasets (OpenOffice, Netbeans, Eclipse, and Mozilla) containing a total of 190,558 BRs, the authors showed that their approach achieves an average F-measure score of 56%-62%. They also showed that Im-ML.KNN improves on average the F-measure scores by 119.69%, 9.11%, and 161.08% when compared with past methods namely the method proposed by Lamkanfi et al. [53], ML.KNN [57], and HOMER-NB [58], respectively.

The authors, however, did not provide a reproduction package, which made it challenging for us to reuse their approach. Reimplementing Im-ML.KNN would require resources and even if we succeeded to do so, it would have been difficult to reproduce their experiments on our datasets because of the number of parameters that we needed to provide, which we could not find (at least explicitly) in the corresponding papers. In addition, the only common dataset between their

approach and ours is the Eclipse dataset.

Despite these challenges, we attempt, in this paper, to provide a preliminary baseline comparison by comparing the results of our approach when applied to the Eclipse BRs with stack traces (this represents only 12.9% of BRs of the entire Eclipse dataset) to the results obtained by Im-ML.KNN applied to the entire Eclipse dataset as reported in their respective papers.

Table IX shows the best F-measures of EnHMM for each BR field and that of Im.ML.KNN. We also measure the improvement. As we can see, although EnHMM is tested on far fewer data points than Im.ML.KNN, the average F-measure score of EnHMM improves the average F-measure score of Im.ML.KNN by 6.80% (this is calculated as follows: (59.78%-55.97)/55.97%).

Table IX. Comparison between EnHMM and Im.ML.KNN based on f-measure

F-measure	**Average**	**Assignee**	**Component**	**OS**	**Priority**	**Product**	**Severity**	**Version**	**Status**
EnHMM	59.78%	87.82%	65.00%	53.83%	63.52%	72.78%	32.66%	66.16%	36.47%
Im-ML.KNN	55.97%	86.67%	63.65%	66.06%	54.13%	73.34%	25.77%	63.41%	14.75%
Improvement	**6.80%**	**1.33%**	**2.12%**	**-18.51%**	**17.35%**	**-0.76%**	**26.74%**	**4.34%**	**147.25%**

EnHMM F-measure score is higher than Im.ML.KNN for five fields out of eight. The major improvements are observed for the “priority”, “severity”, and “status” fields (between 17.35% to 147.25%). Slight improvements can be seen for the “assignee”, “component”, and “version” fields (between 1.33% and 4.34%). For the “OS” field, we observe that EnHMM F-measure score is considerably lower than that of Im.ML.KNN (improvement of -18.51%), possibly because of the low number of reassigned BRs used for training (only 6% as shown in Table VI). This also suggests that having more BRs with stack traces may improve the accuracy of the proposed solution. We intend to conduct more studies to understand the underlying reasons behind the performance of EnHMM across these BR fields. We need to examine in more depth how the size of the dataset,

the quality of the traces, and the use of a particular machine learning algorithm impact the results.

Table X shows a comparison of both approaches using the average precision and recall. Xia et al. [56] did not report the precision and recall obtained by applying Im.ML.KNN to each field. They only included the averages shown in Table X. We can see that, in average, EnHMM has a much higher recall (76.39% compared to 56.13%), but a lower precision (53.93% compared to 56.71%). In other words, EnHMM can predict BR fields better than Im.ML.KNN, but also has a higher false positive rate. We can enhance precision in two ways: (a) add more training BRs with stack traces, and (b) combine other features such as BR field metadata and BR descriptions and summaries (if deemed of good quality).

Table X. Comparison between EnHMM and IM.ML.KNN

Approach	**Average Precision**	**Average Recall**
EnHMM	53.93%	76.39%
Im-ML.KNN	56.71%	56.13%
Improvement	**-4.90%**	**36.09%**

Finding 3:

The average F-measure of EnHMM, trained on 12.9% of Eclipse BRs, outperforms all the reported state-of-art algorithms, which are trained on the entire Eclipse dataset. EnHMM improves the average F-measure by 15% (i.e. 60% from 52%) over im-ML.KNN [56], a leading approach.

4.2.5 Discussion

On the performance of EnHMM: The appealing results obtained by EnHMM are attributable to the power of HMMs in modeling sequential data as opposed to traditional machine

learning techniques, which do not take full advantage of sequential data. Moreover, fusing the weak and best classifiers using 10 different Boolean functions maximizes the diversity between two combined classifiers, in fact, it is the most important ground truth for any ensemble-based approaches [13] [15].

On the use of heterogenous classifiers: EnHMM is based on a combination of multiple HMM homogenous classifiers, trained by varying the number of hidden states. This said, the combination process itself is not linked to the sole use of HMM. It can, for example, be used to combine decisions from other types of classifiers such as those built using SVM, KNN, etc. This can further improve the diversity aspect of the combination process (which is now supported through the use of the Kappa- coefficient).

On the use of stack traces: Our findings clearly show the importance of stack traces in predicting bug report fields. This confirms the need to better collect, store, and manage stack traces whenever a bug report is submitted. For the present time, both Eclipse and Gnome rely on stack traces that are copied and pasted in BR descriptions by end users. This process is error-prone and may result in the presence of noise. Bug report tracking systems must be equipped with powerful mechanisms for managing historical traces that can later be used for all types of applications including the prediction of BR field reassignment.

4.2.6 Limitation

The main limitation of our approach is the low number of BRs that come with stack traces. As an example, only 10% of Eclipse BRs described in [66] contain stack traces. This is because many bug tracking systems are still not equipped with adequate mechanisms for managing traces. Nevertheless, we believe that an approach that uses stack traces remains very useful, especially in

situations where BR descriptions and summaries are deemed to be of poor quality. In addition, our own experience working with industrial partners shows that it is a very common practice in industry to collect stack traces whenever a BR is submitted. This is because traces serve other important purposes such as bug localization and reproduction. We therefore conjecture that, in the future, more bug tracking systems (including those in the open source community) will provide better mechanisms for collecting, storing, and managing stack traces.

4.3 Threats to Validity

Our proposed approach and the conducted experiments are subject to threats to validity, namely external, internal, and construct validity.

Threats to external validity: Our approach is evaluated against two open source datasets. We need to conduct further studies by applying it to more datasets that contain a large number of stack traces to be able to generalize the results. We also need to use other features such as BR descriptions, summaries, and so on to assess the effectiveness of EnHMM on these features in situations where one cannot rely on stack traces.

Threats to internal validity: In our approach, the way we set the hyperparameters A and B, conditional probability matrices, to construct HMM could be a threat to internal validity. We used the validation set to optimize A and B. A different validation set could result in a different initialization of A and B, which my produce another model. However, to our knowledge there is no clear solution to this problem and most studies that use HMM follow random initialization of A and B and repeat this process several times until a satisfactory model is obtained. In addition, we chose to build 40 HMMs for each BR field by varying the number of hidden states. A different configuration may yield other results. Another threat may be with respect to the use of regular

expressions to extract stack traces from BR descriptions. Our regular expression may have missed some stack traces. The missed stack traces could have slightly altered the accuracy of our approach. In addition, we implemented many scripts to extract data, build HMMs, etc. Although care was exercised to write these scripts, errors may have occurred. We will make all our scripts available online to allow other researchers to reproduce our work.

Threats to construct validity: The construct validity shows how the used evaluation measures could reflect the performance of our predictive model. In this study, we used precision, recall, F-measure, ROC curves, and AUC. These measures are widely used in similar studies to assess the accuracy of machine learning models.

4.4 Conclusion

We proposed an effective approach for predicting the reassignment of BR fields. Our approach, called EnHMM, combines multiple HMMs using WPIBC, an anomaly detection algorithm that uses Boolean combination of classifiers, pruned using the Kappa coefficient. When applied to Eclipse and Gnome BR repositories, EnHMM achieves an average precision, recall, and F-measure of 54%, 76, and 56% on Eclipse dataset and 41%, 69%, and 51% on Gnome dataset. A preliminary comparison study shows that EnHMM outperforms leading BR field reassignment prediction methods. Future research should focus on (a) applying EnHMM to larger datasets, (b) understanding the performance of EnHMM by examining the quality of stack traces, (c) combining stack traces with other BR features, and (d) combining other classification techniques, other than HMMs.

Chapter 5. MASKED: A Mapreduce Solution For The Weighted Kappa-pruned Ensemble-based Anomaly Detection System

Detecting system anomalies at run-time is critical for system reliability and security. Studies in this area focused mainly on effectiveness of the proposed approaches; that is, the ability to detect anomalies with high accuracy. However, less attention was given to efficiency. In this paper, we propose an efficient MapReduce Solution for the Kappa-pruned Ensemble based Anomaly Detection System (MASKED). It profiles the heterogeneous features from large-scale traces of system calls and processes them by heterogeneous anomaly classifiers which are Sequence-Time Delay Embedding (STIDE), Hidden Markov Model (HMM), and One-class Support Vector Machine (OCSVM). We deployed MASKED on a Hadoop cluster using the MapReduce programming model. We compared their efficiency and scalability by varying the size of the cluster. We assessed the performance of the proposed approach using the CANALI-WD dataset which consists of 180 GB of execution traces, collected from 10 different machines. Experimental results show that MASKED becomes more efficient and scalable as the file size is increased (e.g., 6-node cluster is 8 times faster than the 2-node cluster). Moreover, the throughput achieved on a 6-node solution is up to 5 times better than a 2-node solution.

5.1. Introduction

Studies have shown that ensemble approaches that combine the decisions of multiple crisp

classifiers[3] using Boolean combination rules such as Pair-wise Brute-force Boolean Combination (BBC2) [10], Iterative Boolean Combination (IBC) [11], and a recently proposed Weighted Pruning Iterative Boolean Combination (WPIBC) [15] improve significantly the detection accuracy, while reducing the false alarms rates which are a major impediment for the general adoption of anomaly detection techniques in practice. Moreover, Wael et al., [23] have shown that a combination of heterogeneous anomaly classifiers (e.g., STIDE [33], OCSVM [80], and HMMs) can significantly improves the overall performance of the system. However, heterogeneous classifiers use heterogeneous features for modeling and testing the normal behavior of a system. For example, OCSVM uses fixed-size vector-based features while HMM and STIDE use fixed-size sliding window-based short sequences of system calls. Therefore, profiling such heterogeneous features from large-scale traces of system calls is the very first and essential step before processing them by the ensemble of heterogeneous anomaly classifiers.

For instance, each trace entry produced by kernel collector [92], contains so many information related to each invoked system call such as arguments, result (return), process ID, process name, parent process ID, etc. Filtering and transforming such a large-scale trace of system calls into numerical sequences of system calls, and then, treating them to profile the heterogeneous features for heterogeneous anomaly classifiers, is a time-consuming task for a single machine. To address this issue, a feasible solution would be to profile the heterogeneous features of the ensemble-based anomaly detection system by leveraging the power of existing parallel computation frameworks, such as HDFS (Hadoop Distributed File System) and the MapReduce programming model which

[3] A crisp anomaly classifier is the one that produces a decision (i.e., normal or anomalous) instead of scores (i.e., likelihood probability or similarity). This is contrasted with a soft classifier, which produces scores instead of a decision. A soft classifier can be converted into one or more crisp anomaly classifiers by setting different thresholds on the output scores [12] [100].

are implemented on Big Data platforms.

However, Hadoop with its original parallel computation model is technically not suitable for profiling sequential data due to dependencies on the temporal information or the orders of a sequence [76]. For example, when HDFS splits a large trace file into two or more fixed-size blocks, Hadoop fails to keep track of the order or temporal information of large sequences within the trace file. To overcome this limitation, Li et. al, [76] have recently proposed an index pool data structure to predict time series by rolling a fixed-size window using Hadoop and the MapReduce programming model. Index pool has shown to be efficient in extracting the index key of a rolling window once the entire sequence is already distributed across multiple splits. However, extracting the index key for each rolling window gives rise to a linear increase of the computational time proportionally to the length of the sequence. Moreover, this approach can only profile the features of sliding windows, and thus, it is not suitable for the ensemble of heterogeneous anomaly classifiers. Therefore, a more sophisticated MapReduce algorithm is required. This algorithm must profile the heterogeneous features such as fixed-size sliding windows for short sequences-based anomaly classifiers (e.g., HMMs and STIDE) and fixed-size feature vectors for the traditional machine learning based anomaly classifiers (e.g., OCSVM).

In this work, we propose an efficient anomaly detection approach called MASKED-A MapReduce Solution for the Kappa-pruned Ensemble-based Anomaly Detection System. MASKED has only one MapReduce job. It profiles the heterogeneous features from the large-scale traces of system calls, and then processes them by a pre-constructed set of Kappa-pruned Ensemble-based Iterative Boolean Combination Rules (BICKER). In constructing BICKER, we use the same technique used in our previous work [15] with the exception of using the input of heterogeneous anomaly classifiers (i.e., multiple HMMs, STIDE, and OCSVM) instead of

homogeneous ones (i.e., only multiple HMMs). BICKER selects a set of diverse soft and their corresponding complementary crisp classifiers which are used to construct the Boolean combination rules. Then, BICKER is used by MASKED to process the profiled heterogeneous features.

The main contributions of this work are as follows:

- Construction of a set of Kappa-pruned Ensemble-based Iterative Boolean Combination Rules (BICKER) by using the WPIBC Boolean combination technique [15]. BICKER takes heterogeneous anomaly classifiers (i.e., multiple HMMs, STIDE, and OCSVM) as input instead of homogeneous ones (i.e., only multiple HMMs) as was the case in WPIBC.

- Selection of five most diverse soft anomaly classifiers (i.e., three HMMs, STIDE, and OCSVM) where each one has six complementary crisp classifiers, which are used to construct the final set of Boolean combination rules.

- A MapReduce Solution for the Kappa-pruned Ensemble-based Anomaly Detection System (MASKED) that profiles heterogeneous features from large-scale traces of system calls and processes them using BICKER.

The rest of this chapter is organized as follows. In Section 5.2, we describe the implementation of our proposed approach followed by the experimental results in Section 5.3. Finally, we conclude the paper in Section 5.4 and discuss the future directions.

5.2 Proposed Approach

In this work, we propose a MapReduce Solution for the Kappa-pruned Ensemble-based

Anomaly Detection System (MASKED) that profiles the heterogeneous features from the large-scale traces of system calls, and then processes them by a pre-constructed set of Kappa-pruned Ensemble-based Iterative Boolean Combination Rules (BICKER). In constructing BICKER, we leverage our previous proposed Weighted Pruning Iterative Boolean Combination (WPIBC) technique [15]. The only difference is that the inputs of BICKER are a set of heterogeneous soft anomaly classifiers (e.g., multiple HMMs, STIDE, and OCSVM) whereas, WPIBC uses homogeneous ones (i.e., only multiple HMMs). BICKER is used by the proposed MapReduce solution (MASKED) to process the profiled heterogeneous features. MASKED is completely controlled by only one MapReduce job that does not only profile the heterogeneous features for the heterogeneous anomaly classifiers (e.g., STIDE, HMM, and OCSVM) but also process them by using BICKER Boolean combination rules. In the following, we first describe the construction procedure of BICKER and then, we present the proposed MapReduce solution.

5.2.1 Kappa-pruned Ensemble-based Iterative Boolean Combination Rules (BICKER)

Although, the construction procedure of BICKER is exactly the same as in WPIBC, the inputs of BICKER are now three main heterogeneous soft anomaly classifiers (STIDE, multiple HMMs, and OCSVM) instead of only homogeneous multiple HMMs. We trained STIDE and HMM using the fixed-size sliding window based sequential features, and OCSVM using the *tf-idf* term vectors (both feature types can be profiled using the proposed MapReduce solution (MASKED) whose details are discussed in the next subsection B). We use the validation set same as in WPIBC [15] for selecting the most diverse soft and their corresponding complementary crisp classifiers.

First, we compute a set of scores for each input soft anomaly classifier. Then, we set all the possible thresholds on each set of scores. Each threshold is associated with a crisp classifier that

produces a set of responses 0/1 (0-means normal and 1-means anomaly), which in turn, produce a single point (*fpr*-false positive rate, *tpr*-true positive rate) on the ROC space. Therefore, each soft classifier produces a set of crisp classifiers or a set of points (*fpr, tpr*) on the ROC space with an AUC (area under the curve) value used as a performance metric for that soft classifier.

With this setting and according to WPIBC [15], we select the most diverse soft classifiers while pruning all the redundant ones using weighted kappa coefficients (an extended version [82] of Cohen's kappa [83] that measures the degree of agreement between two soft classifiers at the various ranks/levels/thresholds). Figure 5.2 shows the selected five diverse base soft classifiers (OCSVM, STIDE, and three HMMs) while pruning 17 redundant soft HMMs.

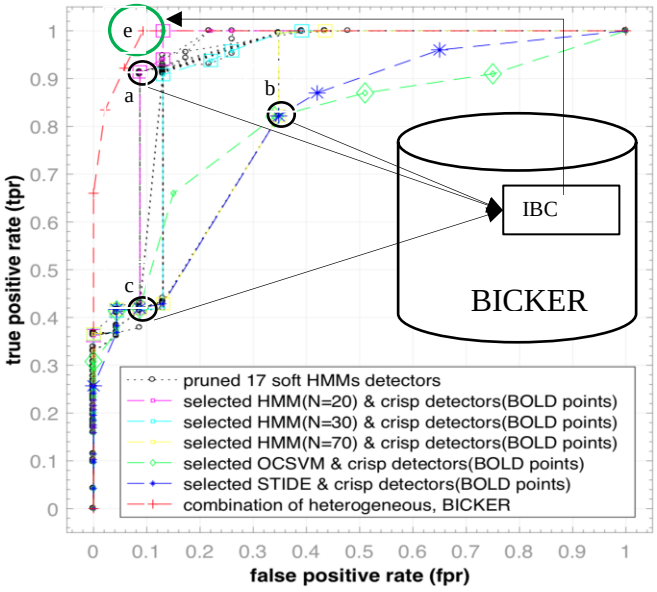


Figure 5.1. Selected diverse heterogeneous soft anomaly classifiers (OCSVM, STIDE, and 3 HMMs) including their corresponding selected complementary crisp classifiers (bold marker points) also using one of the kappa-pruned ensembles based Weighted Pruning Iterative Boolean Combination (WPIBC) techniques [34].

Let the number of possible thresholds be k. Each selected diverse base soft classifier produces k crisp classifiers. Then, we apply the MinMax-kappa pruning technique [12] on each selected soft

classifier. As a result, m (m<<k) complementary crisp classifiers out of k candidate crisp classifiers are selected while the trivial (always produces same responses either 0 or 1) and redundant crisp classifiers are pruned. Figure 5.2 illustrates the selected 6 complementary crisp classifiers (bold marker points) from each selected diverse base soft classifier.

The selected five diverse base heterogeneous soft anomaly classifiers and their corresponding 30 complementary crisp classifiers are then used to construct the final Boolean combination rules. As in WPIBC, we leverage the IBC Boolean combination technique [11] in constructing BICKER. For instance, the ROC curve, red one with '+' marker points (shown in Figure 5.2), is the resulted composite ROC curve using the BICKER Boolean combination rules on the validation set. In Figure 5.2 and for simplicity, we show a composite emerging point (e) which results from the IBC combination of three selected complementary crisp classifiers a, b, and c. The best-case scenario for BICKER is that it uses only the five most diverse base soft classifiers or their selected corresponding 30 complementary crisp classifiers to get this composite ROC curve. In contrast, when IBC is used without pruning, all the available 22 input soft classifiers or 2,200 (in our case, k=100) crisp classifiers should be used to get the same composite ROC curve [11].

Finally, we store the ensuing BICKER information into a NoSQL database: (i) the trained parameters of each selected soft classifiers and the thresholds of their six complementary crisp classifiers, and (ii) the constructed Boolean combination rules using only the selected complementary crisp classifiers. The proposed MapReduce solution that contains only one MapReduce job, uses BICKER for processing the profiled heterogeneous features from a large-scale raw traces of system calls.

5.2.2 Profiling Heterogeneous Features using Distributed File System

It is well known that HDFS, a distributed file system, splits a large file (bigger than the block size, 64MB) into several fixed-size blocks, which are distributed across many parallel nodes [75]. However, if a trace file with a large sequence of system calls is stored into two or more HDFS blocks, the temporal orders of system calls will be lost. That is, some fixed-size sliding windows are straddled at the split boundary between two blocks [76]. Figure 5.4 (a) shows an example in which three consecutive sliding windows (assuming a window of size four): window 8, window 9, and window 10 are straddled at the split boundary between two blocks. Indexing these straddle windows is important for re-assembling them at the aggregation level. In this work, we propose a general solution for indexing these straddle windows, which can be used for profiling both fixed-size sliding window based short subsequences as well as fixed-size feature vectors from a large-scale trace file that is stored in a distributed fashion.

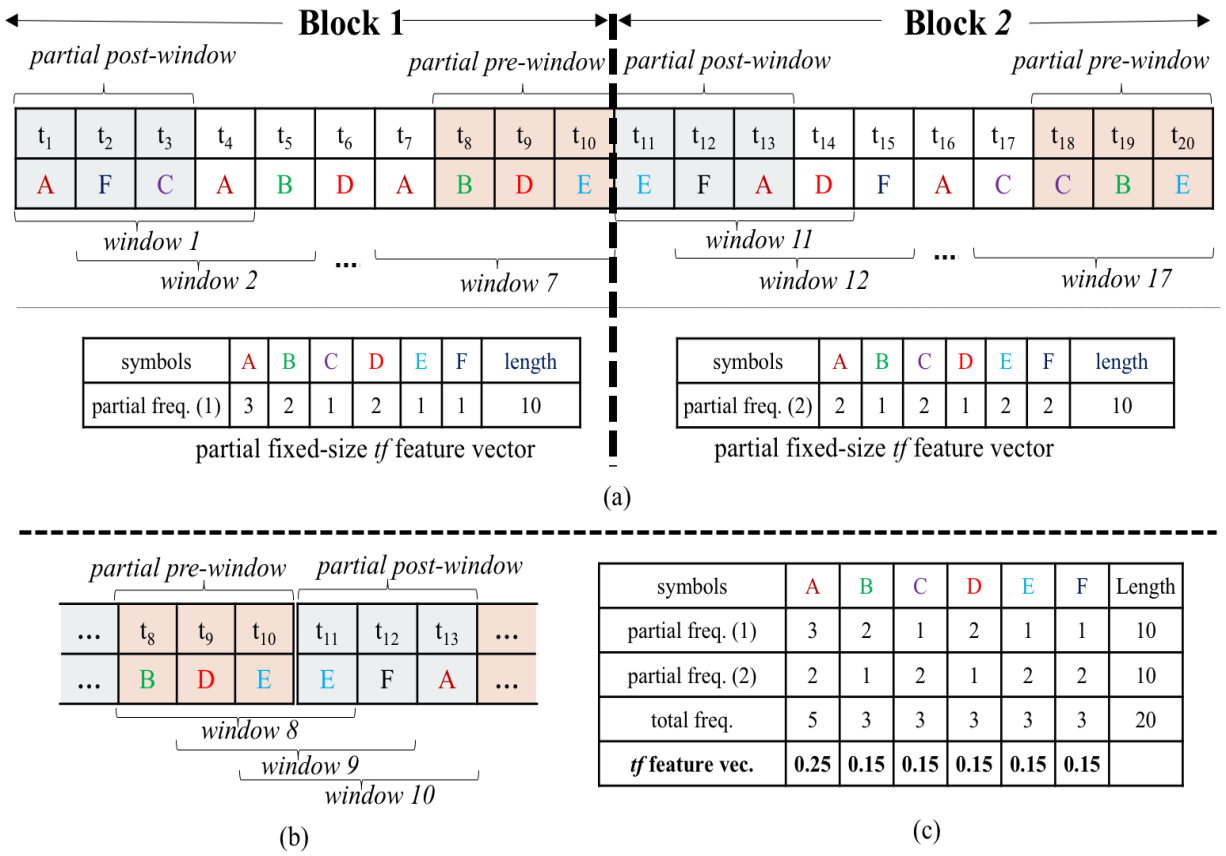


Figure 5.3. A general approach for profiling heterogeneous features from a large-scale trace file that has a long sequence of system calls and stored in a distributed file system

Before profiling the fixed-size sliding windows, each distributed block produces a set of complete sliding windows including two partial windows (partial pre-window and partial post-window) as shown at the top of Figure 5.4 (a). The main benefit of these two partial windows is that, at the aggregation level, only two consecutive partial pre-window and post-window are required to profile the rest of the straddle sliding windows. Figure 5.4 (b) shows that the two-consecutive partial pre-window and post-window are merged into one partial subsequence before being sorted based on the timestamps (t). This partial subsequence is then used to produce the rest of the complete straddle sliding windows (windows 8, 9, and 10) at the split boundary between two blocks.

For profiling the fixed-size feature vector, each block produces a partial *tf* feature vector whose size is fixed and equals the number of unique symbols used in the system. It also records the length of the processed subsequence (within a block) at the end of that partial *tf* vector. The bottom of Figure 5.4 (a) shows two blocks producing two partial *tf* feature vectors with size of six (i.e., the number of unique symbols: A B C D E F), excluding the last element that is the length (10) of the processed subsequence. At the aggregation level, Figure 5.4 (c) shows that the two partial *tf* feature vectors are also merged into a complete *tf* feature vector, normalized by the total length (20) of that sequence. The *tf* feature vector is then transformed into *tf-idf* feature vector using equation (3) and the precomputed document frequency (*df*).

5.2.3 A MapReduce Solution for Profiling and Processing Large-scale Traces of System Calls

In the proposed method, we use a set of system call traces collected by the Anubis emulator tool and stored in HDFS, as a large-scale dataset [84] [92]. Running under Windows operating system, the OS emulator has a kernel module that tracks system call events and annotates them according to privacy rules [92]. Figure 5.6 shows the flow of data of the MapReduce job that extracts, transforms, and profiles the heterogeneous features, and then, processes them using BICKER (as discussed in subsection A).

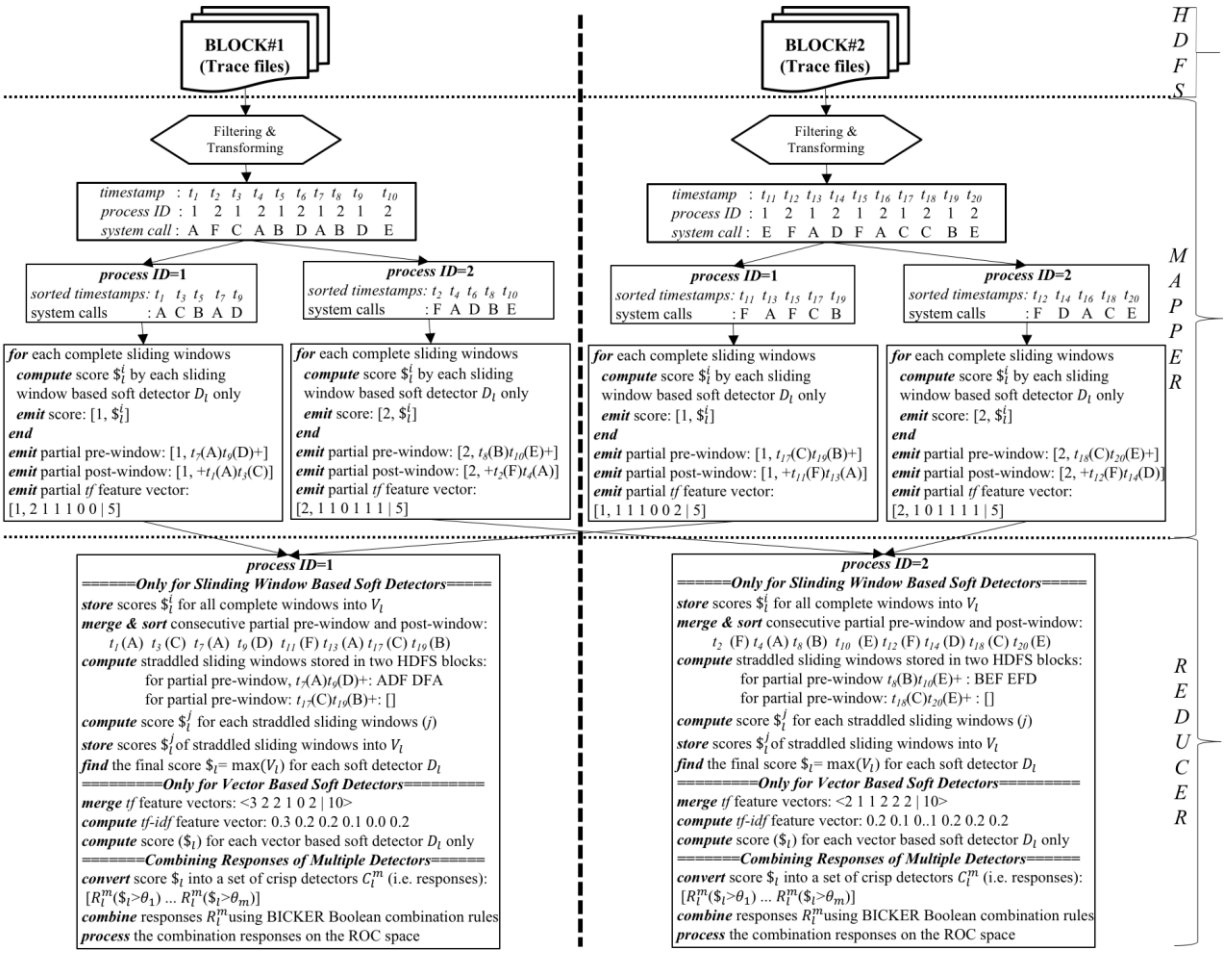


Figure 5.5. The flow of data of the proposed MapReduce solution MASKED for profiling heterogeneous features for heterogeneous anomaly classifiers and processing them using a pre-constructed Kappa-pruned Ensemble based Iterative Boolean Combination Rules (BICKER)

Since each trace file contains so many information related to each invoked system call (e.g., result, pid, process name, and parent process ID), the mapper function first filters and transforms a raw system call trace file into a set of tuples. Each tuple contains three fields: ⟨timestamp, pid, system_call⟩ which are needed to profile the heterogeneous features for the anomaly classifiers. As shown in Figure 5.6, the mapper function groups all the tuples into sub-sequences of system calls based on each process ID. The Mapper function then computes all the complete sliding windows, including the two partial windows. It also computes a partial *tf* feature vector for each

sub-sequence of system calls.

Once a sliding window, w_i is complete, the mapper function accesses BICKER to load the trained parameters of each sliding window based soft classifiers, D_l (e.g., STIDE and three HMMs). Then, it uses them to compute the score $\$_l^i$. The score is then sent as a *key-value* pair into the reduce function, where *key* is the pid and *value* is the score. If the sliding window is partial, the score is not computed, and the partial window is sent as a value together with the pid to the reduce function. Similarly, the mapper function directly sends a partial tf feature vector as a (*key*, *value*) pair into the reduce function, where *key* is the pid and *value* is the partial tf feature vector.

For each process, the reduce function re-assembles (i.e., merges and sorts) the partial windows and uses them to compute the straddled sliding windows which were stored in two HDFS blocks. It also computes the scores ($\$_l^j$) for each straddled sliding windows (w_j) by accessing each sliding window-based soft classifiers (D_l) from BICKER. Then, it aggregates all the scores V_l=[$\$_l^i$ $\$_l^j$] to find the maximum which is considered as the desired score $\$_l$=max($V_l$), for each sliding window based soft classifiers (D_l). Similarly, and for each process, the reduce function aggregates all the partial tf vectors into a single tf vector, normalized with the length of the sequence. The normalized tf vector is further weighted by the document frequency (df) and transformed into tf-idf feature vector using equation (5.3). This tf-idf feature vector is then processed by accessing each vector based soft classifiers, D_l (e.g., OCSVM) from BICKER to compute the score $\$_l$.

The reduce function accesses BICKER to load the thresholds of each soft classifier, D_l, and converts the computed score $\$_l$ into a set of six (m=6) complementary crisp classifiers C_l^m. Then, the responses R_l^m of each crisp classifier C_l^m are combined using the BICKER Boolean rules. Finally, the combination responses R_l^m are used to compute the final composite ROC convex hull

(ROCCH) on the ROC space.

5.3 Experiments and Results

To access the performance of the proposed MapReduce solution, a small cluster with only seven nodes was used as a platform. The CANALI-WD [84] was used as raw traces of system calls dataset.

5.3.1 Setting the Training Parameters

For training, we used the traces of normal behavior of Anubis-good and Goodware datasets (excluding the traces of machine 10, which are used for testing). In addition to the traces of machine 10, malware and malware-test datasets were used to construct the testing set with varied sizes to evaluate the performance of MASKED. Among the evaluation traces, we randomly selected 10% from machine 10, malware, and malware-test datasets to form the validation set. The training dataset was used to train the three-main heterogeneous soft anomaly classifiers (STIDE, multiple HMMs, and OCSVM). In the case of STIDE, we built the normal database using the normal unique short-sequences. We also used the same unique normal short-sequences to train the HMM parameters (A, B, π) using the BW algorithm [3]. In the case of OCSVM, we converted the normal training sequences into the tf-idf feature vectors using Equation (5.3). The converted tf-idf vectors were used to train the OCSVM using the Gaussian or RBS (radial basis function) kernel function [51]. We obtained the best accuracy for OCSVM on the validation set for sigma = 0.001. We obtained the best accuracy for OCSVM soft on the validation set for sigma = 0.001.

To select the best window size for both STIDE and HMM, we trained them with three different window sizes (5, 10, and 20). We obtained the best accuracy using the validation set for a window size of 5 which was selected as the window size. Moreover, to find the well-trained HMMs models,

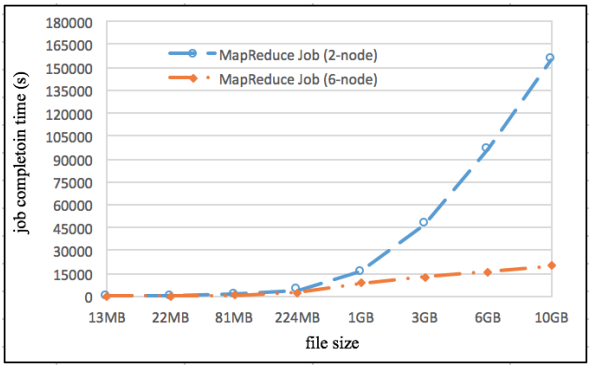
we trained different discrete-time ergotic HMMs with various N values (N=10, 20, … 100) [15].

5.3.2 Cluster Configuration

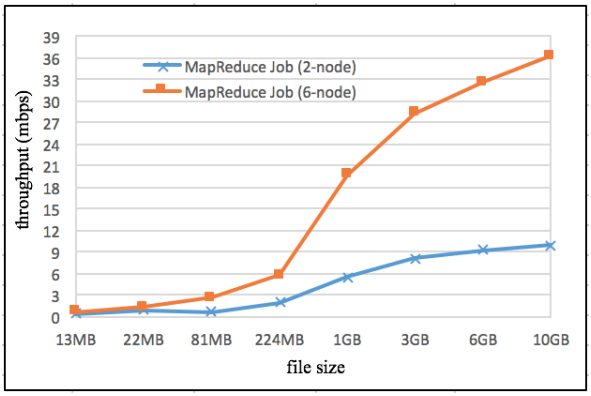
We configured a small Hadoop cluster with only seven nodes to test the proposed approach. We used Matlab Distributed Computing Server [93] to setup this small cluster. Among the seven nodes, six nodes were used as a Hadoop cluster and one node was used as a database server to store the contents of BICKER. The five nodes of Hadoop cluster (excluding the Hadoop master node) were used to accumulate a large-scale system call traces dataset. The Hadoop cluster with six nodes was used as a HDFS with block size of 64MB.

5.3.3 Analyzing Performance of the Proposed MapReduce Solution

We evaluated the performance of the proposed MapReduce solution by varying the input file size from 13MB to 10GB. We compared the performance of 6-node and 2-node Hadoop cluster settings in terms of job completion time (seconds) and throughput (MBps). Figure 5.8 shows the performance of the MapReduce job with different file sizes. According to Figure 5.8 (a), when the file size is very small (up to 81MB), the completion times are almost constant. When the input file size increases above 81MB, however, our approach gave rise to a significant reduction of the completion time with a 6-node cluster compared to 2-node cluster. For example, when the file size is 10GB, the completion times were 20,068s and 155,187s for 6-node and 2-node cluster settings, respectively. That is, MapReduce job with 6-node cluster is approximately 8 times faster than that with 2-node cluster.



(a)



(b)

Figure 5.7. Performance comparison between 6-node and 2-node Hadoop clusters: (a) job completion time and (b) throughput

In terms of throughput and according to Figure 5.8 (b), we can see that when the file size is more than 224MB, the 6-node cluster far outperformed the 2-node cluster. For example, when the file size of 10GB, the 6-node cluster achieved a throughput of 36MBps compared to 9MBps achieved by the, whereas, the 2-node cluster. That is, the throughput of the 6-node cluster is about 4 times higher than that of the 2-node cluster.

We evaluated the scalability of MASKED with the increase of the number of cluster nodes from one node to six nodes. From Figure 5.10, we can see that the MapReduce job reduced the completion time inversely proportional to the number of worker nodes. This result was expected for two reasons: 1) Hadoop is known for its scalability; and 2) MapReduce parallel/distributed computing provides a powerful solution for accessing, processing, grouping, and aggregating a

large-scale data such as the one used in this study.

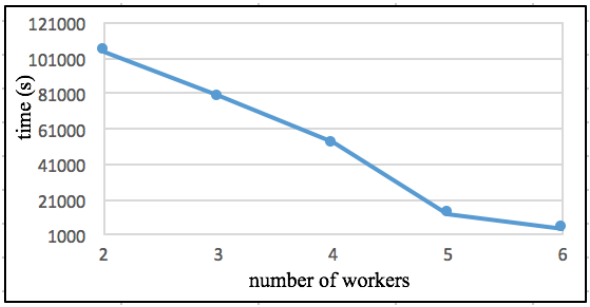


Figure 5.9. Performance comparison with the increase of number of workers, when the file size is fixed to 10 GB.

We analyzed the outputs (i.e., combination responses) of the proposed MapReduce job on the ROC space. Figure 5.12 shows the achieved composite ROCCH (red color) after combining the responses of the selected complementary crisp classifiers. According to this figure, BICKER shows a significant improvement when compared to the performance of the individual classifiers, particularly, when the false alarm is close to zero. These results show conclusively that using heterogeneous classifiers gives rise to better anomaly detection accuracy that using homogeneous multiple HMMs.

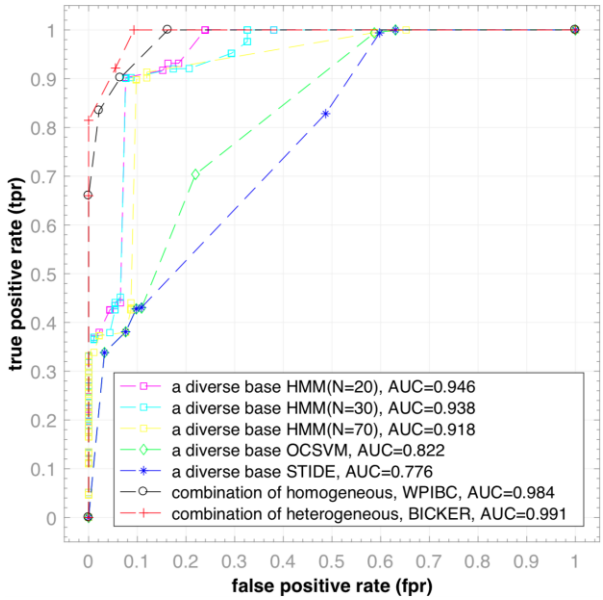


Figure 5.11. Comparing the combination results on the ROC space using the standard AUC (Area Under the Curve) as a measurement metric.

5.3.4 Effects of Partial Pre(Post)-window for Indexing the Straddle Sliding Windows

Instead of using additional indexing data structure, like in [76] [78], the two partial windows are essential for indexing the straddle sliding windows at the split boundary between two HDFS blocks. In contrast, Li method [76] needs to access index pool data structure to profile each complete and partial window.

At the aggregation end, the reduce function only accesses the two consecutive partial windows to produce the remaining complete windows at the split boundary between two blocks (Figure 5.4). In contrast to Li method in which both the mapper and reducer need to maintain many partial windows to produce the remaining complete windows. In addition, the proposed approach does not need to store and access any additional index pool data structure as in [76] [78].

5.3.5 Effects of Heterogeneous Classifiers in Constructing the Boolean Combination Rules, BICKER

It is well-known that the diversity between two combined crisp or soft classifiers is an important factor for any ensemble-based anomaly detection approach [12] [15] [23] [13]. That is, if the responses of two crisp classifiers are comparable, combining them using Boolean combination rules declines the anomaly detection accuracy. In our previous work [15], we developed a Boolean combination approach (WPIBC) which demonstrated how the diversities among the combined soft and crisp classifiers can be guaranteed. This work shows that the diversity is improved when using heterogeneous classifiers. Although the construction of BICKER is exactly the same as in WPIBC, the input is a set of heterogeneous soft anomaly classifiers (STIDE, multiple HMMs, and OCSVM) instead of only homogeneous multiple HMMs. BICKER uses only five diverse base heterogeneous soft anomaly classifiers (3 HMMs, STIDE, and OCSVM) while pruning 17 HMMs classifiers as the redundant ones.

5.4 Conclusion

We proposed an efficient MapReduce solution, namely called MASKED, for the Kappa-pruned Ensemble-based Anomaly Detection Systems. MASKED has only one MapReduce job that profiles the heterogeneous features from large-scale raw traces of system calls for heterogeneous anomaly classifiers. The MapReduce job also processes the profiled heterogeneous features using a constructed kappa-pruned iterative Boolean combination rules, BICKER. The experimental results with varied sizes of HADOOP clusters, have shown that MASKED is efficient and scalable for detecting system anomaly with the help of kappa-pruned ensemble-based anomaly detection system. In the future, we plan to evaluate the efficiency of MASKED with more worker nodes and anomaly classifiers.

To the best of our knowledge, MASKED is the first initiative where MapReduce is used to profile and process the heterogeneous features for heterogeneous anomaly classifiers.

Chapter 6. Conclusions and Future Work

6.1 Conclusions

The main contribution of this thesis is to develop an ensemble of machine learning techniques that selects the most diverse classifiers from a set of input classifiers. We leverage the kappa measure of (dis)agreement to compute the diversities among the set of classifiers. The weighted kappa selects the most diverse soft classifiers. Then, we apply the simple kappa on each diverse soft classifier to find its complementary crisp classifiers. At the end, we leverage Boolean combination techniques to combine the decisions produced by each complementary crisp classifier. We validated the propose solution by applying it to two application domains: detecting system anomalies and detecting bug fields reassignment.

In anomaly detection, we considered two benchmark datasets: ADFA and CANALI. We compared the results with the state-of-art ensemble anomaly detection techniques. The proposed weighted pruning ensemble approach obtained much better results than the other ensemble techniques particularly when the false alarm is almost close to zero. The proposed approach also significantly reduces the number of Boolean operations needed to combine the results from the multiple classifiers because of the fact that it operates on a subset of diverse classifiers only.

Later, we applied the propose weighted pruning ensemble approach in the application of predicting the reassignment of bug report fields, we also considered two different projects: Eclipse and Gnome. For both applications, we compared the results with the state-of-art algorithm.

We further extended the proposed approach by leveraging heterogeneous classifiers and Big Data platforms. We proposed an efficient MapReduce solution, namely called MASKED that

profiles features of heterogeneous diverse anomaly classifiers from large-scale raw traces of system calls. The experimental results with varied sizes of HADOOP clusters, have shown that MASKED is efficient and scalable for detecting system anomaly with the help of kappa-pruned ensemble-based anomaly detection system.

6.2 Future Work

In this thesis, we conducted an extensive research to define the most diverse soft and crisp classifiers from a set of candidate classifiers. We also validated the proposed approach by applying it to a set of homogeneous and heterogeneous classifiers. However, there are potential future directions that would improve the proposed ensemble approach. The following subsections present future directions.

6.2.1 Leveraging Recurrent Neural Networks (RNNs)

As input for the homogeneous candidate classifiers, we use a set of multiple HMMs classifiers. However, recently, deep neural network models are getting more attention because of their high accuracy [94] [95]. One future direction of our research is to train multiple Long Short-term Memory (LSTM) Recurrent Neural Network (RNN) models by varying different learning parameters and use them as input for the proposed WPIBC approach [15]. LSTMs track a long-term dependency over time by analyzing memory states. They are especially best for sequential data as they track dependencies and correlations over time. Since our datasets (both for anomaly detection and bug fields reassignment prediction) are sequential, an ensemble of LSTMs would be a potential future direction for improving the accuracy of the proposed WPIBC approach.

6.2.2 Increasing Diversity

For further improvements, the other potential future direction is to increase the number and type of classifiers. The proposed WPIBC ensures the diversity at the decision level. However, we can also ensure the diversity at the algorithmic level by adding more heterogenous classifiers. Although, we tested WPIBC by leveraging diverse heterogeneous classifiers, our experiments are limited to HMMs, OCSVM, and STIDE. Adding more diverse (and heterogeneous) classifiers such as LSTMs may help in modeling more complex patterns, reducing false positive rates, and also improving the overall accuracy as well. We also increase the diversity of the proposed approach by adding more feature engineering approaches. In this solution, we use sliding windows for training HMMs and STIDE and TF-IDF feature vectors for training OCSVM. However, we may train more accurate and diverse models by leveraging different feature engineering techniques that may capture more complex variations of a system.

6.2.3 Comparing with Other Ensemble Techniques

For further validation and verification of the proposed ensemble approach, we can conduct a comparison study between the proposed and the other existing state-of-art ensemble techniques such as AdaBoost [97] and XGB [98]. We can also verify the soundness of the proposed WPIBC approach by applying it to diverse datasets from various application domains such as datasets from health care systems.

6.3 Closing Remarks

Research shows that the diversity among ensemble of classifiers holds great potential to improve accuracy. In this thesis, we proposed a weighted pruning based Boolean combination technique that ensures the diversity among the combination of classifiers by pruning the redundant

soft classifiers using weighted kappa. We validated successfully the proposed approach by applying it to two application domains in the fields of software security and reliability: detecting system anomalies and predicting the bug report fields reassignment. We hope that this works sets the ground for further research in leveraging artificial intelligence techniques to improve the security and reliability of software systems.

Bibliography

[1] H.-J. Liao, C.-H. R. Lin, Y.-C. Lin and K.-Y. Tung, "Intrusion detection system: A comprehensive review," *Journal of Network and Computer Applications,* vol. 36, no. 1, p. 16–24, 2013.

[2] V. Chandola, A. Banerjee and V. Kumar, "Anomaly Detection: A Survey," *ACM Computing Surveys,* vol. 41, no. 3, pp. 1-58, 2009.

[3] L. E. Baum, T. Petrie, G. Soules and N. Weiss, "A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains," *The Annals of Mathematical Statistics,* vol. 41, no. 1, p. 164–171, 1970.

[4] S. Bhatkar, A. Chaturvedi and R. Sekar, "Dataflow anomaly detection," in *IEEE Symposium on Security and Privacy*, 2006.

[5] Y.-S. Chen and Y.-M. Chen, "Combining incremental hidden Markov model and Adaboost algorithm for anomaly intrusion detection," in *Proceedings of the ACM SIGKDD Workshop on Cyber Security and Intelligence Informatics*, New York, NY, USA, 2009.

[6] W. W. Cohen, "Fast effective rule induction," in *Proceedings of the Twelfth International Conference on Machine Learning*, Tahoe City, California, 1995.

[7] G. Creech and J. Hu, "Generation of a new ids test dataset: Time to retire the kdd collection," in *IEEE Wireless Communications and Networking Conference (WCNC)*, Shanghai, China, 2013.

[8] T. G. Dietterich, "Ensemble methods in machine learning," in *Proceedings of the First International Workshop on Multiple Classifier Systems*, London, UK, 2000.

[9] Y. Du, H. Wang and Y. Pang, "A hidden Markov models-based anomaly intrusion detection method," in *Proceedings of the World Congress on Intelligent Control and Automation (WCICA)*, 2004.

[10] M. Barreno, A. Cardenas and J. D. Tygar, "Optimal roc for a combination of classifiers," in *Proceedings of the 20th International Conference on Neural Information Processing (NIPS)*, 2007.

[11] W. Khreich, E. Granger, A. Miri and R. Sabourin, "Iterative Boolean Combination of Classifiers in the ROC Space: An Application to Anomaly Detection with HMMs," *Journal of Pattern Recognition,* vol. 43, no. 8, pp. 2732-2752, 2010.

[12] A. Soudi and W. H.-L. A. Khreich, "An Anomaly Detection System based on Ensemble of Detectors with Effective Pruning Techniques," in *IEEE International Conference on Software Quality, Reliability and Security*, Aug. 2015.

[13] L. I. Kuncheva, "A bound on kappa-error diagrams for analysis of classifier ensembles," *IEEE Transactions on Knowledge and Data Engineering,* vol. 25, no. 3, pp. 494-501, March 2013.

[14] M. A. Black and B. A. Craig, "Estimating disease prevalence in the absence of a gold standard," *Statistics in Medicine,* vol. 21, no. 18, p. 2653–2669, 2002.

[15] M. S. Islam, W. Khreich and A. Hamou-Lhadj, "Anomaly Detection Techniques Based on Kappa-Pruned Ensembles," *IEEE Transactions on Reliability,* vol. 67, no. 1, pp. 212-229, 2018.

[16] M. S. Islam, K. K. Sabor, A. Hamou-Lhadj, A. Trabelsi and L. Alawneh, "MASKED: A MapReduce Solution for the Kappa-pruned Ensemble-based Anomaly Detection System," in *the 18th IEEE Int. Conf. on Software Quality, Reliability, and Security*, Lisbon, Portugal, 2018.

[17] M. S. Islam, A. Hamou-Lhadj, K. K. Sabor, M. Hamdaqa and H. Cai, *EnHMM: On the Use of Ensemble HMMs and Stack Traces To Predict the Reassignment of Bug Report Fields*, (in preparation).

[18] N. Ebrahimi, A. Trabelsi, M. S. Islam, A. Hamou-Lhadj and K. Khanmohammadi, "An HMM-based approach for automatic detection and classification of duplicate bug reports," *Information and Software Technology, Elsevier,* vol. 113, pp. 98-109, 2019.

[19] N. Ebrahimi, M. S. Islam, A. Hamou-Lhadj and M. Hamdaqa, "An Effective Method for Detecting Duplicate Crash Reports Using Crash Traces and Hidden Markov Models," in *Proc. of the IBM 26th Annual International Conference on Computer Science and Software Engineering (CASCON'16)*, 2016.

[20] J. Kittler, M. Hatef, R. P. W. Duin and J. Matas, "On combining classifiers," *IEEE Transactions on Pattern Analysis and Machine Intelligence* , vol. 20, no. 3, pp. 226-239, Mar 1998.

[21] L. I. Kuncheva, Combining Pattern Classifiers: Methods and Algorithms, Wiley , 2004.

[22] Z. H. Zhou, Ensemble Methods: Foundations and Algorithms, CRC Press. Taylor & Francis Group, 2012.

[23] W. Khreich, S. S. Murtazaa, A. Hamou-Lhadja and C. Talhi, "Combining heterogeneous anomaly detectors for improved software security," *Journal of Systems and Software*, vol. 137, pp. 415-429, February, 2017.

[24] W. Khreich, E. Granger, R. Sabourin and A. Miri, "Combining Hidden Markov Models for anomaly detection," in *International Conference on Communications (ICC)*, Dresden, Germany, June 2009.

[25] T. Fawcett, "An introduction to ROC analysis," *Pattern Recognition Letter,* vol. 27, no. 8, p. 861–874, 2006.

[26] J. Daugman, "Biometric decision landscapes," University of Cambridge Computer Laboratory, United Kingdom, 2000.

[27] Q. Tao and R. Veldhuis, "hreshold-optimized decision-level fusion and its application to biometrics," *Pattern Recognition*, vol. 41, no. 5, p. 852–867, 2008.

[28] S. Haker, W. M. Wells, S. K. Warfield, I.-F. Talos, J. G. Bhagwat, D. Goldberg-Zimring, A. Mian, L. Ohno-Machado and K. H. Zou, "Combining classifiers using their receiver operating characteristics and maximum likelihood estimation," *Medical Image Computing and Computer-Assisted Intervention, Lecture Notes in Computer ScienceLecture Notes in Computer Science, Springer,* vol. 3749, p. 506–514, 2005.

[29] J. Neyman and E. S. Pearson, "On the problem of the most efficient tests of statistical hypotheses," *Philosophical Transactional Royal Society of London A Mathematical Physical and Engineering Science*, vol. 231, p. 289–337, 1933.

[30] P. A. Flach and S. Wu, "Repairing concavities in ROC curves," in *Int. Joint Conf. on Artificial Intelligence*, Edinburgh, Scotland, 2005.

[31] H. Teng, K. Chen and S. Lu, "Adaptive real-time anomaly detection using inductively generated sequential patterns," in *Proceedings of IEEE Computer Society Symposium on Research in Security and Privacy. IEEE Computer Society Press*, 1990.

[32] X. Song, M. Wu, C. Jermaine and S. Ranka, "Conditional anomaly detection," *IEEE Transactions on Knowledge and Data Engineering*, vol. 19, no. 5, p. 631–645, 2007.

[33] S. Forrest, S. A. Hofmeyr, A. Somayaji and T. A. Longstaff, "A sense of self for Unix processes," in *Proceedings of the 1996 IEEE Symposium on Research in Security and Privacy*, Oakland, CA, USA, May 1996.

[34] N. V. Chawla, N. Japkowicz and A. Kotcz, "Editorial: special issue on learning from imbalanced data sets," *ACM SIGKDD Explorations Newsletter*, vol. 6, no. 1, p. 1–6, 2004.

[35] S. S. Murtaza, N. H. Madhavji, A. Hamou-Lhadj and M. Gittens, "Identifying Recurring Faulty Functions in Field Traces of a Large Industrial Software System," *IEEE Transactions on Reliability*, vol. 64, no. 1, pp. 269-283, 2014.

[36] W. Sha, Y. Zhu, M. Chen and T. Huang, "Statistical Learning for Anomaly Detection in Cloud Server Systems: A Multi-Order Markov Chain Framework," *IEEE Transactions on Cloud Computing*, vol. 6, no. 2, pp. 401 - 413, 2018.

[37] A. Bovenzi, F. Brancati, S. Russo and A. Bondavalli, "An OS-level Framework for Anomaly Detection in Complex Software Systems," *IEEE Transactions on Dependable and Secure Computing*, vol. 12, no. 3, pp. 366 - 372, 2015.

[38] J. Yang, X. Du, L. Zhou, S. Shan and B. Cui, "Research on the Identification of Software Behavior in Anomaly Detection," in *10th Int. Conf. on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS)*, 2016.

[39] D. Gizopoulos, M. Psarakis, S. Adve, P. Ramachandran, S. Hari, D. Sorin, A. Meixner, A. Biswas and X. Vera, "Architectures for Online Error Detection and Recovery in Multicore Processors," in *Automation & Test in Europe Conference & Exhibition (DATE)*, 2011.

[40] C. Warrender, S. Forrest and B. Pearlmutter, "Detecting intrusions using system calls: alternative data models," in *Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy*, Oakland, CA, USA, 1999.

[41] S. A. Hofmeyr, S. Forrest and A. Somayaji, "Intrusion detection using sequences of system calls," *Journal of Computer Security,* vol. 6, no. 3, p. 151–180, 1998.

[42] S. Forrest, S. Hofmeyr and A. Somayaji, "The evolution of system call monitoring," in *Computer Security Applications Conference, ACSAC*, Dec 2008.

[43] W. Wang, X.-H. Guan and X.-L. Zhang, "Modeling program behaviors by hidden Markov models for intrusion detection," in *Proceedings of 2004 International Conference on Machine Learning and Cybernetics*, 2004.

[44] L. Rabiner, "A tutorial on Hidden Markov Models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, p. 257–286, 1989.

[45] P. Wang, L. Shi, B. Wang, Y. Wu and Y. Liu, "Survey on HMM based anomaly intrusion detection using system calls," in *5th International Conference on Computer Science & Education*, Hefei, China, Aug. 2010.

[46] D. Y. Yeung and Y. Ding, "Host-based intrusion detection using dynamic and static behavioral models," *Pattern Recognition*, vol. 36, no. 1, p. 229–243, 2003.

[47] X. Zhang, P. Fan and Z. Zhu, "A new anomaly detection method based on hierarchical HMM," in *Proceedings of the Fourth International Conference on Parallel and Distributed Computing, Applications and Technologies*, Chengdu, China, China, 2003.

[48] D. Kang, D. Fuller and V. Honavar, "Learning Classifiers for Misuse Detection Using a Bag of System Calls Representation," in *Lecture Notes in Computer Science*, Berlin, Heidelberg, Springer, 2005, pp. 511-516.

[49] A. Sharmaa, A. K. Pujari and K. K. Paliwal, "Intrusion detection using text processing techniques with a kernel based similarity measure," *Computers & Security,* vol. 26, no. 7-8, pp. 488-495, December 2007.

[50] G. Salton, Automatic text processing: the transformation, analysis, and retrieval of information by computer, Boston, MAUnited States: Addison-Wesley Longman Publishing Co., Inc., January 1989.

[51] C.-C. Chang and C.-J. Lin, "LIBSVM -- A Library for Support Vector Machines," [Online]. Available: https://www.csie.ntu.edu.tw/~cjlin/libsvm/.

[52] A. Iannino and D. J. Musa, "Software Reliability," *Advances in Computers, Elsevier,* vol. 30, pp. 85-170, 1990.

[53] A. Lamkanfi and S. Demeyer, "Predicting reassignments of BRs an exploratory investigation," in *Proc. of the 17th European Conference on Software Maintenance and Reengineering*, 2013.

[54] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj and T. Zimmermann, "What makes a good BR?," in *Proc. of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE'06)*, 2008.

[55] P. J. Guo, T. Zimmermann, N. Nagappan and B. Murphy, "'not my bug!" and other reasons for software BR reassignments," in *Proc. of the Conference on Computer Supported Cooperative Work (CSCW)*, 2011.

[56] X. Xia, D. Lo, E. Shihab and X. Wang, "Automated BR field reassignment and refinement prediction," *IEEE Transactions on Reliability,* vol. 65, no. 3, p. 1094–1113, 2016.

[57] M. L. Zhang and Z. H. Zhou, "Ml-knn: A lazy learning approach to multi-label learning," *Pattern Recognition,* vol. 40, no. 7, p. 2038–2048, 2007.

[58] G. Tsoumakas, I. Katakis and I. Vlahavas, "Effective and efficient multilabel classification in domains with large number of labels," in *Proc. of the Workshop on Mining Multidimensional Data (MMD'08)*, 2008.

[59] Y. Bengio, Markovian Models for Sequential Data, London: Advanced Information and Knowledge Processing. Springer, 2008.

[60] Z. Xing, J. Pei and E. Keogh, "A Brief Survey on Sequence Classification," ACM SIGKDD Explorations Newsletter, 2010.

[61] G. V. Vstovsky and A. V. Vstovskaya, "A class of hidden Markov models for image processing," *Pattern Recognition Letters,* vol. 14, no. 5, pp. 391-396, 1993.

[62] S. Breu, R. Premraj, J. Sillito and T. Zimmermann, "Frequently asked questions in BRs," University of Calgary, Technical Report, 2009.

[63] E. Shihab, A. Ihara, Y. Kamei, W. M. Ibrahim, M. Ohira, B. Adams, A. E. Hassan and K.-I. Matsumoto, "Predicting re-opened bugs: A case study on the eclipse project," in *Proc. of the 17th Working Conference on Reverse Engineering*, 2010.

[64] E. Shihab, A. Ihara, Y. Kamei, W. M. Ibrahim, M. Ohira, B. Adams, A. E. Hassan and K.-I. Matsumoto, "Studying re-opened bugs in open source software," *Springer Journal on Empirical Software Engineering,* p. 1–38, 2012.

[65] A. Sureka, "Learning to classify BRs into components," in *Proc. of the 50th International Conference on Objects, Models, Components, Patterns*, 2012.

[66] K. K. Sabor, A. Hamou-Lhadj and A. Larsson, "DURFEX: A Feature Extraction Technique for Efficient Detection of Duplicate BR," in *Proc. of the IEEE International Conference on Software Quality, Reliability and Security (QRS'17)*, 2017.

[67] J. Lerch and M. Mezini, "Finding duplicates of your yet unwritten BR," in *Proc. of the 17th European Conference on Software Maintenance and Reengineering (CSMR'13)*, 2013.

[68] K. K. Sabor, M. Nayrolles, A. Trabelsi and A. Hamou-Lhadj, "An Approach for Predicting BR Fields Using a Neural Network Learning Model," in *Proc. of the IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, 2018.

[69] K. K. Sabor, M. Hamdaqa and A. Hamou-Lhadj, "Automatic prediction of the severity of bugs using stack traces," in *Proc. of the 26th Annual International Conference on Computer Science and Software Engineering (CASCON)*, 2016.

[70] K. K. Sabor, M. Hamdaqa and A. Hamou-Lhadj, "Automatic prediction of the severity of bugs using stack traces and categorical features," *Elsevier Journal on Information and Software Technology,* vol. 123, 2020.

[71] K. K. Sabor, A. Hamou-Lhadj, A. Trabelsi and J. Hassine, "Predicting BR fields using stack traces and categorical attributes," in *Proc,of the 29th Annual International Conference on Computer Science and Software Engineering (CASCON '19)*, 2019.

[72] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Communications of the ACM,* vol. 51, no. 1, p. 107–113, 2008.

[73] W. Hadley, "The split-apply-combine strategy for data analysis," *Journal of Statistical Software*, vol. 40, no. 1, p. 1–29, 2011.

[74] "Apache Hadoop," [Online]. Available: http://hadoop.apache.org/.

[75] IBM, "What is the Hadoop Distributed File System (HDFS)?," 2014. [Online]. Available: www.ibm.com/software/data/infosphere/hadoop/.

[76] L. Li, F. Noorian, D. Moss and P. Leong, "Rolling window time series prediction using MapReduce," in *Proceedings of the 2014 IEEE 15th Int. Conf. on Information Reuse and Integration (IEEE IRI 2014)*, 2014.

[77] S. Matthews and A. S. Leger, "Leveraging MapReduce and Synchrophasors for Real-Time Anomaly Detection in the Smart Grid," *IEEE Transactions on Emerging Topics in Computing*, vol. 7, no. 3, pp. 392-403, 2017.

[78] Z. Li, F. Hu, J. L. Schnase, D. Q. Duffy, T. Lee, M. K. Bowen and C. Yang, "A spatiotemporal indexing approach for efficient processing of big array-based climate data with MapReduce," *Int. Journal of Geographical Information Science*, vol. 31, no. 1, pp. 17-35, 2017.

[79] H. Kim, J. Kim and I. Kim, "Behavior-based anomaly detection on big data," in *proceedings of the 13th Australian Information Security Management Conference*, 2015.

[80] W.-H. Chen, S.-H. Hsu and H.-P. Shen, "Application of SVM and ANN for intrusion detection," *Computers & Operations Research*, vol. 32, no. 10, pp. 2617-2634, October 2005.

[81] Y. Liao and V. Vemuri, "Use of K-Nearest Neighbour classifier for intrusion detection," *Computers & Security*, vol. 21, no. 5, pp. 439-448, 2002.

[82] C. Valiquette, A. Lesage and C. Mireille, "Computing Cohen's Kappa coefficients using SPSS MATRIX," *Behavior Research Methods, Instruments, & Computers*, vol. 26, no. 1, pp. 60-61, 1994.

[83] J. Cohen, "A coefficient of agreement for nominal scales," *Educational & Psychological Measurement*, vol. 20, p. 37–46, 1960.

[84] D. Canali, A. Lanzi, D. Balzarotti, C. Kruegel, M. Christodorescu and E. Kirda, "A quantitative study of accuracy in system call-based malware detection," in *Proceedings of the 2012 International Symposium on Soft- ware Testing and Analysis, ISSTA 2012, ACM*, NY, USA, 2012.

[85] A. Agresti, Categorical Data Analysis, New York: Wiley, 1990, p. 367.

[86] M. Egele, T. Scholte, E. Kirda and C. Kruegel, "A survey on automated dynamic malware-analysis techniques and tools," *ACM Computing Surveys*, vol. 44, no. 2, March 2008.

[87] S. Hangal and M. S. Lam, "Tracking down software bugs using automatic anomaly detection," in *Proc. of the 24th Int. Conf. on Software Engineering (ICSE)*, ACM, NY, USA, 2002.

[88] A. L. Goel and K. Okumoto, "Time-Dependent Error-Detection Rate Model for Software Reliability and Other Performance Measures," *IEEE Transactions on Reliability,* Vols. R-28, no. 3, pp. 206 - 211, 1979.

[89] L. D. Fosdick and L. J. Osterweil, "Data Flow Analysis in Software Reliability," *ACM Computing Surveys,* vol. 8, no. 3, pp. 305-330, Sept. 1976.

[90] A. Pramod, T. Krishnaprasad, M. Surendra, S. Amit and B. Tanvi, "Understanding City Traffic Dynamics Utilizing Sensor and Textual Observations," in *Proc. of the 13th AAAI Conf. on Artificial Intelligence*, 2016.

[91] N. Bettenburg, R. Premraj, T. Zimmermann and S. Kim, "Extracting structural information from BRs," in *Proc. of the International Working Conference on Mining Software Repositories (MSR'08)*, 2008.

[92] A. Lanzi, M. Christodorescu, D. Balzarotti, E. Kirda and C. Kruegel, "AccessMiner: Using System-Centric Models for Malware Protection," in *Proc. of the 17th ACM Conference on Computer and Communications Security*, 2010.

[93] MATLAB, "MathWorks," [Online]. Available: https://www.mathworks.com/help/matlab-parallel-server/configure-a-hadoop-cluster.html.

[94] K. Gyuwan, Y. Hayoon, L. Jangho, P. Yunheung and Y. Sungroh, "LSTM-Based System-Call Language Modeling and Robust Ensemble Method for Designing Host-Based Intrusion Detection Systems," *ArXiv, Cryptography and Security,* vol. abs/1611.01726, 2016.

[95] D. Kwon, H. Kim, J. Kim, S. C. Suh, I. Kim and K. J. Kim, "A survey of deep learning-based network anomaly detection," *Cluster Computing, Springer,* vol. 22, p. 949–961, 2017.

[96] T. K. Ho, "Random Decision Forests," in *Proceedings of the 3rd International Conference on Document Analysis and Recognition*, Montreal, QC, August 1995.

[97] Y. Freund and R. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of Computer and System Sciences,* vol. 55, p. 119–139, 1997.

[98] T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, San Francisco, CA, USA, 2016.

[99] M. Stamp, "A Revealing Introduction to Hidden Markov Models," Dec 11, 2015.

[100] W. Khreich, E. Granger, A. Miri and R. Sabourin, "Boolean combination of classifiers in the ROC space," in *20th International Conference on Pattern Recognition*, Istanbul, Turkey, 2010.

[101] B. Gao, H.-Y. Ma and Y.-H. Yang, "HMMs (Hidden Markov Models) based on Anomaly Intrusion Detection Method," in *Proceedings of 2002 International Conference on Machine Learning and Cybernetics*, 2002.

[102] "Anubis," [Online]. Available: http://anubis.iseclab.org,2011.