Chapter - 5

DYNAMIC LEARNING CLASSIFIER FRAMEWORK (DLCF) FOR IDS
Chapter 5

DYNAMIC LEARNING CLASSIFIER FRAMEWORK (DLCF) FOR IDS

In this chapter we investigate the task of classification performed by intrusion detection expert and present a novel concept of building a Dynamic Learning Classifier Framework (DLCF) using machine learning techniques. We analyze the requirements of such a system, select a suitable machine-learning technique and validate the system on real kddcup99 dataset.

5.1 DLCF Framework

The architecture of DLCF for IDS is shown in figure 5.1. The DLCF can be implemented either in expert mode or automation mode. In a conventional setup, alarms generated by IDSs are passed onto an IDS expert analyst. The analyst uses his or her knowledge to distinguish between false and true positives and to understand the severity of the alarms.

Conventional systems may use manual knowledge engineering to build an alarms classifier or may use no alarms classifier at all. In any case, the conventional setup does not take advantage of the fact that the analyst is analyzing the alarms in real-time: the manual knowledge of engineering is separated from analyzing alarms.

As shown in Figure 5.1, our system classifies alarms and passes them to the IDS expert. It also assigns a classification confidence (or confidence for short), to alarms, which shows the likelihood of alarms belonging to their assigned classes.
The IDS expert reviews this classification and reclassifies alarms, if necessary. This process is recorded and used as training by the machine learning component to build an improved alarm classifier.

Currently we use a simple human-computer interaction model, where the IDS expert explicitly classifies alarms into true and false positives. In addition to the training examples, we use background knowledge to learn improved classification rules. These rules are then used by DLCF to classify alarms. The expert can inspect the rules to make sure they are correct.

The architecture presented describes the operation of the system in IDS expert mode. The second mode, automation mode, introduces autonomous processing to reduce the operator’s workload.

5.1.1 DLCF in Expert Mode

In expert mode Figure 5.1(a), DLCF classifies alarms and passes all of them to the console to be verified by the IDS expert. In other words, the system assists the IDS expert suggesting the correct classification. The advantage for the IDS expert is that each alarm is already pre-classified and that the IDS expert has only to verify its correctness. The IDS expert can prioritize his or her work, e.g., by dealing with alarms classified as true positives first or sorting the alarms by classification confidence. It is important to emphasize that at the end, the analyst will review all classifications made by the system.

It has been used as a simple human-computer interaction model, in which the expert sequentially classifies alarms into true and false positives, which are converted into training examples; however, more sophisticated interaction techniques are also possible.
Figure 5.1  DLCF Framework in expert and automation modes
More formally, there is a human intrusion detection IDS expert O reviewing a sequence of intrusion detection alarms $(A_1, A_2, \ldots, A_i, \ldots)$ in the alarm log $L$. The review is done by assigning one of the predefined set of classes $\{C_1, C_2, \ldots, C_n\}$ (which can be in particular two classes: true positives and false positives {“+”, “-”}) to each alarm.

The review is typically done sequentially and in real-time, which means that alarm $A_{i+1}$ is reviewed only after alarms $(A_1, A_2, \ldots, A_i)$ have been reviewed and, at this time, alarms $(A_{i+2}, \ldots)$ are not known. This procedure is shown in figure 5.2

**Given** – A sequence of alarms: $(A_1,A_2, \ldots,A_i, \ldots)$ in the alarm log $L$,

a set of classes $C = \{C_1,C_2, \ldots,C_n\}$,

an intrusion detection IDS expert O sequentially and in real-time assigning classes to alarms,

a utility function $U$ minimizing the misclassification cost,

**Find** A classifier classifying alarms, maximizing the utility function $U$.

**Figure 5.2 Assigning Class Labels to Alarms in Expert Mode**

This is a conventional incremental learning setup. Figure 5.3 shows the operation of DLCF in the expert mode. The function goodClassificationPerformance() estimated the performance of the classifier on a confusion matrix using a weighted accuracy (WA) with a threshold $WA_{th}$.
In the automation mode, the high level goal uses a modified utility function $U$, so that it also limits IDS expert’s workload and therefore a system that autonomously processes some alarms is preferred over the one that does not. More formally it is shown in figure 5.4

Input: a sequence of alarms $(A_1, A_2, \ldots, A_n)$
Result: a sequence of classified alarms $((A_1, C_{A_1}), (A_n, C_{A_n}), \ldots, (A_n, C_{A_n}))$

```plaintext
1  initialize;
/* alarms used for the initial training */
2  $x \leftarrow x^0$;
3  while $x < n$ do
4    $S_i \leftarrow$ subsequence($A_1, \ldots, A_x$);
5    $C_i \leftarrow$ learnUpdateClassifier($C_{i-1}, S_i$);
6      while goodClassificationPerformance(WAt) do
7        $C_x \leftarrow$ classify($C_i, A_x$);
8        $C_{Ax} \leftarrow$ askIDS expertVerifyClassification($A_x, C_x$);
9        updateClassificationPerformance($C_x, C_{Ax}$);
10       $x \leftarrow x + 1$;
11      end
12    $i \leftarrow i + 1$
13  end
```

**Figure 5.3 DLCF Classification in Expert Mode**

### 5.1.2 DLCF in Automation Mode

In the automation mode, the high level goal uses a modified utility function $U$, so that it also limits IDS expert’s workload and therefore a system that autonomously processes some alarms is preferred over the one that does not. More formally it is shown in figure 5.4
In Automation Mode the DLCF autonomously processes some of the alerts based on criteria defined by the expert (i.e., classification assigned by DLCF and classification confidence).

By processing alerts we mean that DLCF executes user-defined actions associated with the class labels and classification confidence values. For example, attacks classified as false positives can be automatically removed, thus reducing the analyst’s workload. In contrast, alerts classified as true positives and successful attacks can initiate an automated response, such as reconfiguring a router or firewall.

It is important to emphasize that such actions should be executed only for alerts classified with high confidence, whereas the other alerts should still be reviewed by the analyst. The operation of DLCF in automation mode is shown in figure 5.5.

Note that autonomous alarm processing may change the behavior of the system and negatively impact its classification accuracy. To illustrate this with an example, suppose the system classifies alarms into true and false positives and it is configured to autonomously discard the latter if the classification confidence is higher than a given threshold value.

Given – A sequence of alarms: \( \{A_1, A_2, \ldots, A_n, \ldots\} \) in the alarm log \( L \),
a set of classes \( C = \{C_1, C_2, \ldots, C_n\} \),
an intrusion detection IDS expert \( O \) sequentially and in real-time assigning classes to alarms,
a utility function \( U \) minimizing the misclassification cost,

Find A classifier classifying alarms, maximizing the utility function \( U \).

Fig. 5.4 Classifying Alarms in Automation Mode
Suppose the system learned a good classifier and classifies alarms with high confidence. In this case, if the system starts classifying all alarms as false positives then these alarms would be autonomously discarded and would never be seen by the IDS expert.
These alarms would not become training examples and would never be used to improve the classifier. Another problem is that alarms classified and processed autonomously cannot be added to the list of training examples as the IDS expert has not reviewed them. If alarms of a certain class are processed autonomously more frequently than alarms belonging to other classes (as in the above example), as a consequence we change the class distribution in the training examples.

This has important implications as machine-learning techniques are sensitive to class distribution in training examples. In the optimal case, the distribution of classes in training and testing examples should be identical.

To alleviate these problems, we use a technique called random sampling. In this technique we randomly select a fraction $s$ of alarms which would normally be processed autonomously and instead forward them to the IDS expert. This ensures the stability of the system. The value of $s$ is a tradeoff between how many alarms will be processed autonomously and how much risk of misclassification is acceptable.

5.2 **DLCF Model learning based on RIPPER**

Among the machine learning techniques that best fulfill our requirements, we chose RIPPER [102] a fast and effective rule learner. It has been successfully used in intrusion detection (e.g., on system call sequences and network connection data [47] as well as related domains and it has proved to produce concise and intuitive rules.

As reported by Lee [42], RIPPER rules have two very desirable conditions for intrusion detection: good generalization accuracy and concise conditions. Another advantage of RIPPER is its effectiveness with noisy datasets.
RIPPER has been well documented in the literature, however, for the sake of a better understanding of the system we will briefly explain how RIPPER works. As shown in Algorithm 4, RIPPER learns a sequence RS of rules Ri in the form:

\[ \text{if (condition1 and condition2 and ... conditionN) then class.} \]

A single condition is in the form \( A_i = v \) (in the case of categorical attributes) or \( A_i >= 0 \) or \( A_i <= 0 \) (in the case of numerical attributes). The rule evaluates to true if and only if all its conditions hold, in which case, the prediction is made and no further rules are evaluated.

In a multi-class setting, RIPPER sorts the classes \( C_1, C_2, \ldots, C_n \) in increasing frequency and induces the rules sequentially from the least prevalent class (SC1) to the second to last most prevalent class (SC\(_{n-1}\)). The most prevalent class SC\(_n\) is called a default class, for which no rules are induced. Hence, in the binary case, RIPPER induces rules only for the minority class.

The process of inducing rules for a single class proceeds in two stages: the building stage and the optimization stage. In the building stage, RIPPER builds the rules in the following two steps: growing and pruning. In the growing step, rules are greedily “grown” by adding conditions that maximize the information gain [56]. In the pruning step, rules are pruned using a criterion, which is equivalent to precision. The goal of pruning is to improve both the generalization and the simplicity of the rule. In the optimization stage, building and pruning is executed on both an initial rule and an empty rule set, with the evaluation done on the entire rule-set. Finally, the best variant of the two is selected for the final rule-set.
Unfortunately, the standard RIPPER algorithm is not cost-sensitive and does not support incremental learning. We used the following methods to circumvent these limitations.

5.2.1 Cost-Sensitive Modeling

In a cost-insensitive world, both types of misclassifications (false negatives and false positives) carry equal weights and hence the performance of a classifier can be evaluated by means of accuracy. However, in the real-world the costs of misclassifications are most often not equal, e.g., missing an intrusion is intuitively more expensive than investigating one false positive. This, together with the fact that cost-sensitive problems are typically skewed, increases an importance of cost-sensitive modeling.

In general, cost-sensitive modeling is a difficult issue [20] as there can be many costs that need to be taken into account. For example, Fan [21] defines two types of costs in the domain of intrusion detection: the damage cost $D_{Cost}$, which characterizes the maximum amount of damage inflicted by an attack and a response cost $R_{Cost}$, which is the cost to take action when a potential intrusion is detected.

In this case, false negatives would incur the $D_{Cost}$ for the given attack, false positives and true positives would incur $D_{Cost}$ for the given attack and the wrongly identified attacks would incur both the response cost for the action taken and the damage cost of the missed attack. Moreover, the damage and the response costs are typically not constant and depend on both the attack class and in some cases, the particular instance of an attack (e.g., the damage incurred as a result of an attack against an important server is typically much higher than for the same attacks against a workstation, which can simply be switched off).
In addition, Fan showed that certain features used for testing have different costs than others, e.g., analyzing a flag Transmission Control Protocol (TCP) header is much “cheaper” in terms of resources than calculating statistics over an entire TCP flow. The approach proposed by Fan allows taking this fact into account in building ensemble-based learning systems.

However, while this approach is correct in the formal sense, it has two main problems. First, Fan used boosting methods, in which misclassified instances are “penalized” according to the misclassifications the weak learner made.

While taking both RCost and DCost into account can be easily achieved in this iterative learning method, as a side effect it produces a number of weak classifiers, which make their rules less interpretable. For example with 200 boosting rounds, each of them building a classifier producing 50 rules, there would be 10000 rules that would need to be investigated. In contrast, our approach focuses on a single classifier. Second, the multi-cost sensitive approach introduces a high number of parameters that would need to be investigated.

5.2.2 Binary vs. Multi-Class Classification

It has been seen the job of an intrusion detection analyst and possible classifications of intrusions. Here, we argue that our setup, in which the human analyst analyzes alerts generated by IDS, can be without loss of functionality considered a binary classification problem. First, if multiple classes are used, they are not very systematic and, in most cases, describe a nature of a problem, which either is uniquely determined by the type of IDS alerts at question (e.g., an PORTSCAN alert if it is a true positive is a “scanning incident”), or cannot be determined with certainty.
This means that in many cases, such a classifier, knowing that an alert is a true positive, can be built as a second-stage classifier, or should not be built at all. Second, the costs of misclassifying a certain type of an intrusion as another one are extremely hard to determine.

However, the actual cost of misclassifying different types of alerts as non-attacks is not identical. To illustrate this with an example, missing a scanning incident is much less costly than missing a single stealthy attack that installs a root-kit on a machine.

However, the problem is that those “cheap” attacks are fairly easy to identify and moreover, they constitute a large numbers of alerts. Conversely stealthy attacks are much more difficult to detect (that is why they are called “stealthy”).

This problem of redundancy in the data stream can be solved in two ways: First, alert correlation systems aim at reducing the redundancy in the alert stream and the number of alerts passed to the analyst.

Second, we propose to assign a weight to alerts normalizing them so that the costs of missing different attacks would be identical. This weight should be a function of an alert type so that with n categories of alerts, only n parameters would have to be estimated.

In our evaluation, as we wanted to evaluate minimizing the number of parameters that need to be set, we decided not to take this approach and assumed that the cost of missing all attacks is identical.
5.2.3 Cost-Sensitive RIPPER

As the base version of RIPPER is cost-insensitive, we had to adapt it to support misclassification costs. Among the various methods of making a classification technique cost-sensitive, we focused on those that are not specific to a particular machine-learning technique. By changing costs appropriately, these methods can also be used to address the problem of skewed class distributions. These methods produce comparable results, although this can be data dependent [6]. Experiments not documented here showed that in our context Weighting gives better run-time performance, MetaCost most likely because of the learning of multiple models by. Therefore we chose Weighting for our system.

Weighting re-samples the training set so that a standard cost-insensitive learning algorithm builds a classifier that optimizes the misclassification cost. The input parameter for Weighting is a cost matrix, which defines the costs of misclassifications for individual class pairs.

5.3 DLCF Evaluation

In this section we evaluate DLCF, our Dynamic Learning Classification System presented in Chapter 5. Recall that DLCF can operate in two modes: (i) the expert mode, in which alarms are classified and forwarded to the IDS expert and (ii) the automation mode, which in addition to the classification allows for a fraction of alarms to be processed automatically (e.g., false positives can be discarded) without IDS expert’s intervention. In this section we will verify the operation of DLCF in both these modes. In particular, we would like to test the following two hypotheses:

Hypothesis 5.3.1: The proposed background knowledge improves the accuracy of alarm classification.
Hypothesis 5.3.2: DLCF has acceptable false-positive and false-negative rates in both recommender and automation modes and is useful for intrusion detection.

For the evaluation, the following remark is in place: While evaluating the performance of any binary classifier (or alarm-classification system in particular), we characterize its performance by its confusion matrix and the terms: true positives, false positives, false negatives and true negatives. This causes conflict with terms false positives, true positives commonly used in the domain of intrusion detection and referring to the classification of alarms. In fact, an IDS is a special type of a binary classifier and these names are justified.

To avoid confusion, in the remainder of this dissertation we use terms false negatives, true positives and false positives only in the context of the evaluation of alarm-classification systems.

From now onwards it has been refer to the original classification of alarms as true alarms and false alarms.

5.4 Evaluation Methodology

The evaluation of supervised components of our system is performed in a streamline fashion classifying alarms sequentially as they would be seen by the human IDS expert. We purposely did not use standard machine learning evaluation techniques using stratified cross-validation, because the streamline method better reflects the way the system would be used in practice.

In fact, the system leverages the dependency between alarms by its incremental nature: Misclassified alarms are used to learn an improved alarm classifier and classify future similar alarms correctly.
In the evaluation we use ROC analysis to determine the influence of background knowledge and set system parameters. Subsequently, we evaluate false-negative (FN) and false-positive (FN) rates. We also plot evaluation charts showing how these rates vary during system’s runtime (as a function of classified alarms) and evaluate the overall cumulative numbers.

5.5 Results Obtained with DARPA 1999 Data Set

We evaluated the performance of DLC in expert and automation mode.

5.5.1 Results of DLCF in Expert mode.

In expert mode the IDS expert reviews each alarm and corrects DLCF misclassifications. We plotted the number of misclassifications: false positive rate (Figure 5.6) and false-negative rate (Figure 5.7) as a function of processed alarms. Note that we cropped high error rates at beginning of the run. These are transient effects and we are interested in the asymptotic values.

The resulting overall false-negative rate (fn = 0.024) is much higher than the false-negative rate for the batch classification on the entire dataset (fn = 0.0076). At the same time, the overall false-positive rate (fp = 0.025) is less than half of the false-positive rate for batch classification (fp = 0.06).

These differences are expected due to different learning and evaluation methods used, i.e., batch incremental learning vs. 10-fold cross-validation. Note that both DLCF and a batch classifier have very good classification accuracy and yield comparable results in terms of accuracy.
In automation mode DLCF processes alarms autonomously based on criteria defined by the IDS expert, described in Section 5.1. We configured the system to forward to the IDS expert all alarms classified as true alarms and those false alarms that were classified with low confidence (confidence < cth). The system discarded all other alarms, i.e., false alarms classified with high confidence, except for a fraction s of randomly chosen alarms, which were also forwarded to the IDS expert.
Similarly to the expert mode, we calculated the number of misclassifications made by the system. We experimented with different values of $c_{th}$ and sampling rates $s$. We then chose $c_{th} = 90\%$ and three sampling rates $s$: 0.1, 0.25 and 0.5.

Our experiment is shown in figure 5.7 that the sampling rates below 0.1 make the agent misclassify too many alarms and significantly changes the class distribution in the training examples. On the other hand, with sampling rates much higher than 0.5, the system works similarly to expert mode and is less useful for the IDS expert.

Notice that there are two types of false negatives in automation mode, the ones corrected by the IDS expert and the ones the IDS expert is not aware of because the alarms have been discarded.

Figure 5.7  False positives for DLCF in Automation Mode
We plotted the second type of misclassification as mirrored series with no markers in Figure 5.4a. Intuitively with lower sampling rates, the agent will have fewer false negatives of the first type, in fact missing more alarms. As expected the total number of false negatives is lower with higher sampling rates.

One is surprised to observe that the recommender and the agent have similar false positive rates and similar false-negative rates, even with low sampling rates.

This seemingly counterintuitive result can be explained if we note that automatic processing of alarms classified as false positives effectively changes the class distribution in training examples in favor of true alarms. As a result the agent performs comparably to the recommender.

As shown in Figure 5.8, with the sampling rate of 0.25, more than 60% of false alarms were processed and discarded by DLC. At the same time the number of unnoticed false negatives is half the number of mistakes for expert mode. Our experiments show that the system is useful for intrusion detection IDS experts as it significantly reduces the number of false positives with fairly good accuracy.

The results were particularly clear for the DARPA 1999 data set. We showed that the system is useful in recommender mode, where it dynamically learns the classification from the expert. For dataset we obtained false-negative and false positive rates comparable to batch classification. Note that in recommender mode all system misclassifications are corrected by the expert. In addition, it is found that our system is useful in automation mode, where some alerts are autonomously processed.
The results were particularly clear for the DARPA 1999 data set. We showed that the system is useful in recommender mode, where it dynamically learns the classification from the expert. For dataset we obtained false-negative and false positive rates comparable to batch classification. Note that in recommender mode all system misclassifications are corrected by the expert. In addition, it is found that our system is useful in automation mode, where some alerts are autonomously processed (e.g., false positives classified with high confidence are discarded).
More importantly, for kddcup’99 dataset the false-negative rate of our system is comparable to that in the recommender mode. With this real dataset the system reduced the number of false positives by 60% with a false-negative rate below 0.026 (half of these alerts would have been shown to the analyst) and a false-positive rate 0.025.

5.6 Chapter Summary

In our evaluation of DLCF carried on real-time dataset, DARPA 1999 dataset and I validated the.

The results were particularly clear for the DARPA 1999 data set. We showed that the system is useful in recommender mode, where it dynamically learns the classification from the expert. For dataset we obtained false-negative and false positive rates comparable to batch classification. Note that in recommender mode all system misclassifications are corrected by the expert. In addition, it is found that our system is useful in automation mode, where some alerts are autonomously processed (e.g., false positives classified with high confidence are discarded).

More importantly, for kddcup’99 dataset the false-negative rate of our system is comparable to that in the recommender mode. With this real dataset the system reduced the number of false positives by 60% with a false-negative rate below 0.026 (half of these alerts would have been shown to the analyst) and a false-positive rate 0.025.