Mining in a Data-flow Environment: Experience in Network Intrusion Detection[1]

Wenke Lee   Salvatore J. Stolfo   Kui W. Mok
Computer Science Department
Columbia University
500 West 120th Street, New York, NY 10027
{wenke,sal,mok}@cs.columbia.edu

PaperID: 414

Abstract

In this paper we discuss the KDD process in "data-flow" environments, where unstructured and time dependent data can be processed into various levels of structured and semantically-rich forms for analysis tasks. Using network intrusion detection as a concrete application example, we describe how to construct models that are both *accurate* in describing the underlying concepts, and *efficient* when used to analyze data in real-time. We present procedures for analyzing frequent patterns from lower level data and constructing appropriate features to formulate higher level data. The features generated from various levels of data have different computational costs (in time and space). We show that in order to minimize the time required in using the classification models in a real-time environment, we can exploit the "necessary conditions" associated with the low-cost features to determine whether some high-cost features need to be computed and the corresponding classification rules need to be checked. We have applied our tools to the problem of building network intrusion detection models. We report our experiments using the network data provided as part of the DARPA Intrusion Detection Evaluation program, and our experience in using the mined models in NFR, a real-time network intrusion detection system.

Keywords: data cleaning, feature construction, model costs, intrusion detection.

Contact Author: Wenke Lee, wenke@cs.columbia.edu, (212) 939-7078.

[1]This research is supported in part by grants from DARPA (F30602-96-1-0311) and NSF (IRI-96-32225 and CDA-96-25374).

1 Introduction

In many business and engineering applications, raw data collected from fielded systems needs to be processed into various forms of structured and semantically-rich records before data analysis tasks can produce accurate, useful and understandable results. Consider the problem of credit card fraud detection. Each credit card purchase transmits a stream of data containing the credit card number, amount, and the type of merchandise, etc., to the credit card company's data server. The data is then processed into a "transaction record" that contains additional computed features (fields) measuring the short and long term activity behavior of the account. In network intrusion detection, a packet "sniffer" such as *tcpdump* [JLM89] can be used to record each passing network packet. The data then needs to be processed into "connection records" that contains for each connection its hosts, service (e.g., *telnet*, *ftp*, etc.), and other features that describe the network states. We refer to such applications that involve real-time data collection, processing, and analysis as "data flow" environments.

The *accuracy* of data analysis depends not only on the algorithms employed, but to a very large degree, on the quality of the processed data. For example, if the activity history of the account, e.g., "number of X type of purchases in the past n hours", which is very useful in predicting certain frauds, is missing from the credit card transaction records, the accuracy of the fraud detection system will suffer. However defining such features, that is, determining the constituent attributes of records, have traditionally been based on deep domain knowledge. In a real-time environment, the *efficiency* of data analysis is critical. For example, on-line fraud detection systems need to respond, with an approval or rejection on the transaction, within a few seconds. A trade-off between model accuracy and model evaluation efficiency needs to be considered.

Researchers have recognized that KDD is not a single plan of well-defined operations, rather, it is a sequence of iterative steps that include (but not limited to) data cleaning and preprocessing, feature extraction and data mining, and consolidating and utilizing the discovered knowledge [FPSS96]. However, most of the research in KDD focuses on the data mining step, which assumes the availability of preprocessed data. Since KDD is about finding useful *knowledge*, it would be interesting to see whether and how we can apply KDD techniques to automate some of the knowledge-intensive data preprocessing tasks. Many data mining techniques consider only accuracy and/or complexity measures as evaluation criteria when extracting models from data. The problem of efficiency, i.e., the cost, in time and space, of model execution in real-time has not been adequately studied.

To address these important and challenging problems adequately, one needs to have access to the "data flow" in its entirety. This is sometimes impossible because of legal or organizational constraints. Our research is in a somewhat different and advantageous context since, in applying data mining techniques to build network intrusion detection models [LSM98, LSM99], data is abundant nearly everywhere (all one needs is a computer on a network). In this paper, we focus our discussion on the automatic techniques of comparing frequent patterns mined from normal and intrusion data, and constructing appropriate features for building classification models. We report the results of the 1998 DARPA Intrusion Detection Evaluation, which showed that the detection models produced using our data mining programs performed better than or equivalent to the knowledge engineered intrusion detection systems. We also discuss techniques of deriving the very low cost "necessary" conditions of each learned rule, so that unnecessary (i.e., wasteful) rule checking in real-time execution can be avoided.

Our contributions to KDD are: new techniques for feature construction based on frequent patterns mined from the data with a "minimum" preprocessing; a simple yet useful pattern encoding and comparison technique to assist model construction; strategies for minimizing the cost of model execution in real-time; and most importantly, the demonstration (i.e., through objective evaluation) of the feasibility and advantage of applying these more systematic and semi-automatic data mining approaches to network intrusion detection, an important, challenging and traditionally knowledge engineering application area.

This rest of the paper is organized as follows. We first give a brief overview of the problem of network intrusion detection, and summarize the extentions we made to the basic association rules [AIS93] and frequent episodes [MTV95] algorithms, that are based on the characteristics of network audit data. We then discuss the feature construction steps, which include mining and comparing two sets of patterns to identify the "intrusion only" frequent patterns, and parsing each such pattern to construct temporal statistical features. We report the experiments of using our techniques on the DARPA data. We then describe how to utilize the associations between low cost features and the class labels as the "necessary" conditions for rules to be checked in real-time execution.

2 Data Mining and Intrusion Detection

Intrusions are actions that aim to compromise the security goals of a computer system, namely, confidentiality, integrity, and availability. Intrusion detection (ID) is considered an integral part of critical infrastructure protection mechanisms. Traditionally, intrusion detection systems (IDSs) have been built using purely knowledge engineering approaches, that is, system builders encode their security knowledge into detection models. The manual and ad hoc nature of the development process impinges upon the effectiveness and adaptability of IDSs (in the face of new attack methods or changed network configurations).

We aim to automate the process of building IDSs as much as possible. We are developing a framework, described in detail in [LSM99], of applying data mining techniques to build intrusion detection models. This framework consists of classification and meta-classification [CS93] programs, association rules and frequent episodes programs, as well as a feature construction system. The end product is concise and intuitive classification rules that can detect intrusions.

In [LSM98], we discussed and demonstrated the need to select and construct a set of temporal and statistical features in order to build accurate classification models to detect network intrusions. For example, since a large number of "rejected" network connections in a very short time span is strong evidence of some intrusions, we need to include a feature that measures this indicator.

We proposed to use the frequent sequential patterns mined from audit data as guidelines for feature construction [LSM98]. The process of using data mining approaches to build intrusion detection models is shown in Figure 1. Here raw (binary) audit data is first processed into ASCII network packet (or host event data), which is in turn summarized into connection records (or host session records) containing a number of within-connection features, e.g., *service*, *duration*, *flag* (indicating the normal or error status according to the protocols), etc. Data mining programs are then applied to the connection records to compute the frequent sequential patterns, which are in turn analyzed to construct additional features for the connection records. Classification programs, for example, RIPPER [Coh95], are then used to inductively learn the detection models. This process is of course iterative. For example, poor performance of the classification models often

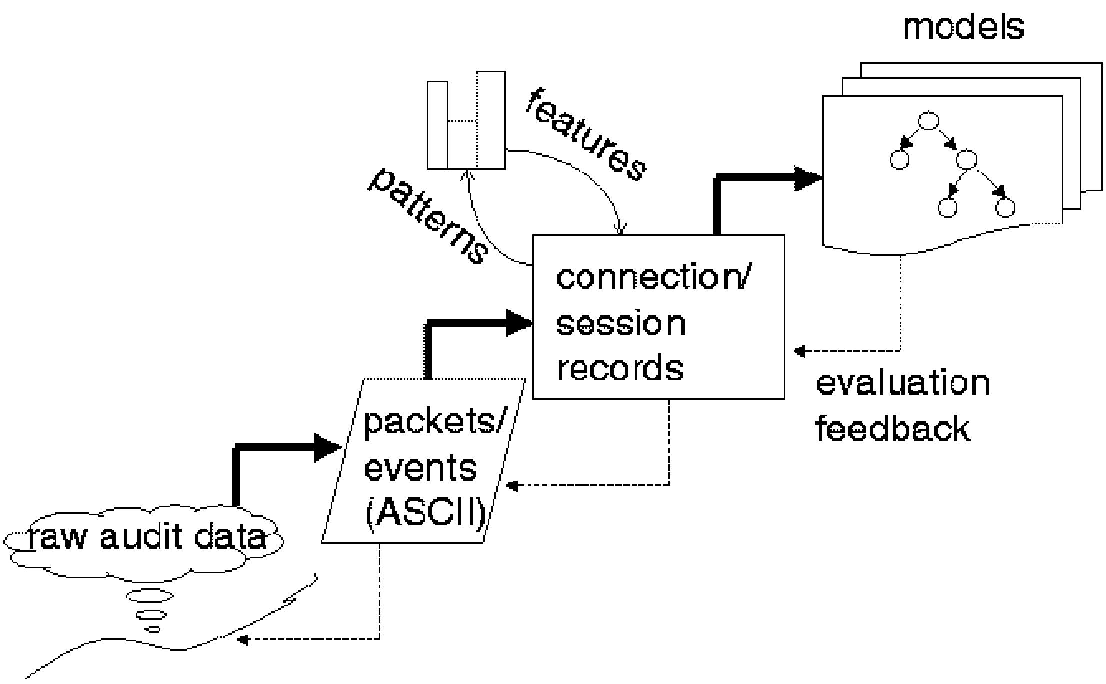


Figure 1: The Data Mining Process of Building ID Models

indicates that more pattern mining and feature construction is needed.

2.1 Mining Audit Data

It is important to incorporate domain knowledge to direct data mining algorithms to find "relevant" patterns efficiently. We described in detail several extensions to the basic association rules and frequent episodes algorithms that utilize the schema-level information about connection records in mining frequent patterns [LSM98, LSM99]. We briefly describe the main ideas here.

Observe that a network connection can be uniquely identified by the combination of its $time$ (start time), src_host (source host), src_port (source port), dst_host (destination host), and $service$ (destination port), which are the "essential" attributes of network data. We argue that "relevant" association rules should describe patterns related to the essential attributes. Depending on the objective of the data mining task, we can designate one (or several) essential attribute(s) as the *axis* attribute(s) [LSM98], which is used as a form of item constraint in association rule mining. During candidate generation, an item set must contain value(s) of the axis attribute(s). When axis

time	duration	service	src_host	dst_host	src_bytes	dst_bytes	flag	...
1.1	0	http	spoofed_1	victim	0	0	S0	...
1.1	0	http	spoofed_2	victim	0	0	S0	...
1.1	0	http	spoofed_3	victim	0	0	S0	...
1.1	0	http	spoofed_4	victim	0	0	S0	...
1.1	0	http	spoofed_5	victim	0	0	S0	...
...	...	...	...	...	...	...	...	...
10.1	2	ftp	A	B	200	300	SF	...
13.4	60	telnet	A	D	200	2100	SF	...
...	...	...	...	...	...	...	...	...

Table 1: Network Connection Records

attributes are used, the frequent episodes algorithm first finds the frequent associations about the axis attributes, and then computes the frequent sequential patterns from these associations. Thus, the associations among attributes and the sequential patterns among the records are combined into a single rule.

Some essential attributes can be the *references* of other attributes. These reference attributes normally carry information about some "subject", and other attributes describe the "actions" that refer to the same "subject". For example, if we want to study the sequential patterns of connections to the same destination host, then dst_host is the "subject" and $service$ is the action. When *reference* attribute [LSM99] is used, the frequent episodes algorithm ensures that, within each episode's minimal occurrences, the records covered by its constituent item sets have the same reference attribute value. An example frequent episode derived from the connection records in Table 1, with *service* as the axis attribute and dst_host as the reference attribute, is shown in Table 2. This is a pattern of the "syn flood" attack, where the attacker sends a lot of "half-opened" connections (i.e., $flag$ is "S0") to a port (e.g., "http") of a victim host in order to over-run its buffer and thus achieve "denial of service".

Frequent episode	Meaning
$(service = http, flag = S0)$, $(service = http, flag = S0)$ $\rightarrow (service = http, flag = S0)$ $[0.93, 0.03, 2]$	93% of the time, after two *http* connections with $S0$ flag are made (to a host *victim*), within 2 seconds from the first of these two, the third similar connection is made, and this pattern occurs in 3% of the data

Table 2: Example Frequent Episode Rule

3 Feature Construction from Mined Patterns

When packet data is summarized into the connection records (see Figure 1) using commonly available packet processing engines, each record contains a set of "intrinsic" features that are for general network traffic analysis purposes. These features are: *service*, *src_host*, *src_port*, *dst_host*, *dur* (duration of the connection), and *src_bytes* and *dst_bytes* (number of data bytes from each direction). The frequent sequential patterns from these initial connection records can be viewed as statistical summaries of the network activities. Therefore, by comparing the patterns from a "normal" dataset (e.g., collected from normal network traffic over an extended period of time) and an "intrusion dataset" (e.g., from recorded simulation runs of attack programs), we can identify and utilize the "intrusion only" patterns for feature construction.

Our experience showed that the choice of axis and reference attributes is very important in computing the intrusion patterns. For example, "port-scan" is an intrusion where the attacker typically makes connections to many ports (i.e., using many different *services*) of a host in a short period of time. A lot of such connections will be "rejected" since many ports are normally closed. Using dst_host as both the axis and reference attribute produces very distinct intrusion patterns, for example, $(flag = REJ, dst\_host = host_A) \rightarrow (flag = REJ, dst\_host = host_A)$. But no intrusion pattern is found when using the *service* as the axis attribute and *dst_host* as the reference attribute since a large number of different services are attempted in a short period time, and as a result, for each service the "same destination host connection rejected" sequential patterns are not frequent.

We need to alleviate the user from the burden of "guessing" these choices. An iterative procedure that involves pattern mining and comparison, feature construction from patterns, and model building and evaluation, is thus employed. In each iteration, a different combination of axis attribute and reference attribute is selected. The choices are limited among the essential attributes, that is, $service$, dst_host, src_dst, or src_port. Note that the exact $time$ is never frequent and is thus omitted. Since intrusions are generally targeted to some victim host(s) in the network, we start with $service$ as the axis attribute and dst_host as the reference attribute. For each iteration, the set of features along with the performance of the resulting classifier, in both TP (true positive) and FP (false positive) is recorded. The set of features that results in the best model is selected at the end of this procedure. We focus our discussion on pattern comparison and feature construction here.

3.1 Pattern Comparison

In order to create "baseline" normal patterns to compare against frequent patterns from intrusion datasets, we mine patterns from each subset (e.g., each day) of normal network connection records, and incrementally *merge* the patterns to an aggregate pattern set [LSM98]. This is done for each possible combination of axis and reference attributes.

The aggregate normal pattern set is usually very large, in the range of several thousands to tens of thousands of patterns. We have developed an *encoding* scheme to convert each frequent pattern to a number so that we can easily visualize (and thus understand), and efficiently compare the patterns. We first encode the associations because they are the constituent itemsets of frequent episodes (due to the use of axis attribute, see Section 2).

The goal of our encoding scheme is to map associations that are structurally and syntactically more "similar" to closer numbers. We also seek an encoding scheme that is simple to compute and manipulate. To define the "similarity" measure precisely, we first define a partial order on all the discovered associations. Assuming the records have n attributes, we call an association $(A_1 = v_1, A_2 = v_2, ..., A_k = v_k)$ "complete and ordered" if $k = n$ and attributes $A_1, A_2, ..., A_k$ are in some user-defined decreasing "order of importance" (e.g., in the simplest form, this can be the alphabetical order of the attribute names). A discovered association can always be converted to its "complete and ordered" form by first inserting $A_i = null$ for each "missing" attribute A_i, and then sorting the attributes in the order of importance. For two "complete and ordered" associations, we say $(A_1 = v_1, A_2 = v_2, ..., A_n = v_n) < (A_1 = u_1, A_2 = u_2, ..., A_n = u_n)$, if $v_j = u_j$, for $j = 1, 2, ..., i - 1$, and $v_i < u_i$. We say association X_i is more "similar" to X_j than to X_k if $X_i < X_j < X_k$ (or $X_k < X_j < X_i$) holds.

Given a set of associations, we use the following algorithm to compute the encodings:

- Convert each association to its "complete and order" form.

- The encoding of an association $(A_1 = v_1, A_2 = v_2, ..., A_n = v_n)$ is a number $e_{v_1} e_{v_2} \ldots ev_n$. That is, the order of the digits, from most to least significant, corresponds to the decreasing "order of importance" of the attributes.

- Each e_{v_i} is:
  - 0 if v_i is $null$, i.e., attribute A_i is missing from the original association;
  - the order of appearance of v_i among all the values of A_i seen (processed) thus far in the encoding process. (Other forms of ordering, for example, based on some user-defined criteria, can be trivially incorporated.)

association	encoding
$(flag = SF, service = http, src\_bytes = 200)$	11001
$(service = icmp\_echo, dst\_host = host_B)$	02100
$(flag = S0, service = http, src\_host = host_A)$	21010
$(service = user\_app, src\_host = host_A)$	03010
$(flag = SF, service = icmp\_echo, dst\_host = host_B, src\_host = host_C)$	12120
...	...

Table 3: Encodings of Associations

When encoding associations from network records, we use the following decreasing "order of importance": *flag*, axis attribute, reference attribute, the rest of "essential" attributes in alphabetical order, and the remaining attributes in alphabetical order. The attribute *flag* is most important (i.e., interesting) in an association since its value is a summary of how the connection has behaved according to the network protocols. Any value other than "SF" (i.e., normal connection establishment and termination) is of great interest for intrusion detection. Table 3 shows some examples of encodings. Here $service$ is the *axis* attribute, and *dst_host* is the reference attribute. The associations are encoded (processed) in the order of their positions in the table (i.e., first row first).

An advantage of our encoding scheme is that we can use simple arithmetic operations to very easily control the "level of detail" required for analysis or comparison of the associations. For example, if we choose to "ignore" src_bytes, we can simply do an integer division of 10 on the encodings.

With the encodings of associations, we can now map an episode rule, $X, Y \rightarrow Z$, where X, Y, and Z are associations (itemsets), to a 3-d data point $(encoding_X$, $encoding_Y$, $encoding_Z)$ for pattern visualization. Due to the difficulties of manipulating n-dimensional ($n > 3$) displays, for a "longer" episode rule $L_1, L_2, ..., L_i \rightarrow R_1, R_2, ..., R_j$ $(i, j \geq 1)$, we use its general (subsuming) form $L_1, L_2 \rightarrow R_j$.

For pattern comparison, we first convert the 3-d encoding of an episode into a 1-d value. Assume $endcoding_X = x_1x_2...x_n$, $endcoding_Y = y_1y_2...y_n$, and $endcoding_Z = z_1z_2...z_n$, then the 1-d representation is $x_1z_1y_1x_2z_2y_2 \ldots x_nz_ny_n$. This presentation preserves the "order of importance" of attributes (in association encoding) and considers the rule structure of an episode. Here two episodes that have similar first "body" (i.e., X) and "head" (i.e., Z) will be mapped to closer numbers. As an example, using the association encodings in Table 3 (and "ignoring" dst_host, src_host, and src_bytes), the "syn flood" pattern in Table 2 is encoded as 222111. Similarly, a "normal" pattern, $(flag = SF$, $service = http)$, $(flag = SF$, $service = icmp\_echo) \rightarrow (flag = SF$, $service = http)$, is encoded as 111112.

When comparing two episodes using their 1-d numbers, a simple digit-wise comparison is performed. That is, in the resulting *diff* score, $d_{x_1}d_{z_1}d_{y_1}d_{x_2}d_{z_2}d_{y_2} \ldots d_{xn}d_{zn}d_{yn}$, each digit, e.g., d_{x_i}, is the absolute value difference in the corresponding digit, e.g., x_i, of the two episodes. For example, when comparing the "syn flood" pattern with the "normal" pattern, the *diff* score is 111001.

Given the normal patterns and patterns from an intrusion dataset, that are computed using the same choices of axis attribute(s), reference attribute(s), support, confidence, and window requirements, we can identify the intrusion only patterns using the following procedure:

- Encode all the patterns;
- For each pattern from the intrusion dataset, calculate its *diff* score with each normal pattern; keep the lowest *diff* score as the "intrusion" score for this pattern;
- Output all patterns that have non-zero "intrusion" scores, or a user-specified top percentage of patterns with the highest "intrusion" scores. For example, since there is no normal pattern with $flag = S0$ in all its itemsets, the *diff* score for the "syn flood" pattern, e.g., 111001, is very high, and thus the pattern will be selected.

This procedure considers a pattern from the intrusion dataset as "normal" as long as it has a match with one of the normal patterns. For simplicity, we omit the comparisons on the support and confidence values (once the heads and bodies of the rules match). We have not seen a case where

Input: a frequent episode, and the set of existing features in connection records, $\mathcal{F}$
Output: the updated $\mathcal{F}$
Begin
(1) Let F_0 (e.g., dst_host) be the reference attribute used to mine the episode;
(2) Let w, in seconds, be the minimum width of the episode;
    /* all the following features consider only the connections in past w
     * seconds that share the same value in F_0 as the current connection
     */
(3) Let $count\_same_{F_0}$ be the number of these connections;
(4) $\mathcal{F} = \mathcal{F} \cup \{count\_same_{F_0}\}$;
(5) **for** each "essential attribute" F_1 other than F_0 **do begin**
(6)   **if** the same F_1 value is in all the item sets **then begin**
(7)   Let $precent\_same_{F_1}$ be the percentage of connections that share the same F_1 value as the current connection;
(8) $\mathcal{F} = \mathcal{F} \cup \{percent\_same_{F_1}\}$;
        end else
        /* there are different F_1 or no F_1 values at all */
(9)   Let $percent\_diff_{F_1}$ be the percentage of different F_1 values in the connections;
(10) $\mathcal{F} = \mathcal{F} \cup \{percent\_diff_{F_1}\}$;
        end
    end
(11) **for** each value V_2 of an "non-essential" attribute F_2 **do begin**
(12)   **if** V_2 is in all the item sets **then begin**
(13)   Let $percent\_same_{V_2}$ be the percentage of connections that share the same V_2 value as the current connection;
(14) $\mathcal{F} = \mathcal{F} \cup \{percent\_same_{V_2}\}$;
(15)   **end else if** F_2 is a numerical attribute **then begin**
(16)   Let $average_{F_2}$ be the average of the F_2 values of the connections;
(17) $\mathcal{F} = \mathcal{F} \cup \{average_{F_2}\}$;
        end
    end
end

Figure 2: Constructing Features from Frequent Episode

two matched rules have values that are more than 5% apart from each other, which is considered an acceptable threshold.

3.2 Feature Construction

Each of the intrusion only patterns (e.g., the "syn flood" pattern shown in table 2) is used for constructing additional features into the connection records, using the algorithm in Figure 2. This procedure parses a frequent episode and uses three simple operators, $count$, $percent$, and $average$, to construct simple statistical features. These features are also temporal since they measure only the connections that are within a time window w and share the same reference attribute value. The intuition behind the feature construction algorithm comes from the straightforward interpretation of a frequent episode. For example, if the same attribute value appears in all the itemsets of an episode, then there is a large percentage of records (i.e., the original data) that have the same value. We treat the "essential" and "non-essential" attributes differently. The "essential" attributes describe the anatomy of an intrusion (a sequence of connections), for example, "the same $service$ (i.e., $port$) is targeted". The actual values, e.g., "http", is often not important because the same attack method can be applied to different targets, e.g., "ftp". On the other hand, the actual "non-essential" attribute values often indicate the invariant of an intrusion regardless of the actual targets, e.g., $flag$ is "S0", because they summarize the behavior according to the network protocols.

As an example, the "syn flood" pattern results in the following additional features: a count of connections to the same dst_host in the past 2 seconds, and among these connections, the percentage of those that have the same $service$ as the current, and the percentage of those that have the "S0" $flag$.

An open problem here is how to decide the right time window value w. Our experience shows that when we plot the number of patterns generated using different w values, the plot tends to stabilize after the initial sharp jump. We call the smallest w in the stable region w_0. Our experiments showed that the plot of accuracies of the classifiers that use the temporal and statistical features calculated with different w, also stabilizes after $w \geq w_0$ and tend to taper off. Intuitively, a requirement for a good window size is that its set of sequential patterns is stable, that is, sufficient patterns are captured and noise is small. We therefore use w_0 for adding temporal and statistical features.

3.3 Experiments with DARPA data

We participated in the 1998 DARPA Intrusion Detection Evaluation Program, prepared and managed by MIT Lincoln Labs. The objective of this program is to survey and evaluate research in intrusion detection. A standard set of extensively gathered audit data, which includes a wide variety of intrusions simulated in a military network environment, was provided by DARPA. Each participating site was required to build intrusion detection models or tweak their existing system parameters using the training data, and send the results (i.e., detected intrusions) on the test data back to DARPA for performance evaluation. We summarize our experience here[2].

We were provided with about 4 gigabytes of compressed raw (binary) *tcpdump* data of 7 weeks of network traffic, which can be processed into about 5 million connection records, each with about 100 bytes. The two weeks of test data have around 2 million connection records. Four main categories of attacks were simulated: DOS, denial-of-service, e.g., syn flood; R2L, unauthorized access from a remote machine, e.g., guessing password; U2R, unauthorized access to local superuser (root) privileges, e.g., various of "buffer overflow" attacks; and PROBING, surveillance and probing, e.g., port-scan.

Using the procedures discussed in Section 3, we compared the aggregate normal pattern set with the patterns from each dataset that contains an attack type. (The intrusion data set is created by extracting the connection records that fall within a surrounding time window of plus and minus 5 minutes of the whole duration of an attack. The training data contains "list files" specifying the timestamps and participating hosts and ports of each intrusion.) The following features were constructed according to the intrusion only patterns:

- The "same host" features that examine only the connections in the past 2 seconds that have the same destination host as the current connection, and calculate statistics related to protocol behavior, service, etc.
- The similar "same service" features that examine only the connections in the past 2 seconds that have the same service as the current connection.

We call these the (time-based) "traffic" features of the connection records. There are several "slow" PROBING attacks that scan the hosts (or ports) using a much larger time interval than 2 seconds, for example, one in every minute. As a result, these attacks did not produce intrusion

[2]Full detail about features constructed for the DARPA data set appear in a companion paper [LSM99]. We summarize our experiments here so that the paper is self-contained. Also it is possible that the DARPA dataset used in this study may be the subject of the KDD Cup and hence for fairness to all participants, we wish not to provide the detail here.

only patterns with a time window of 2 seconds. We sorted these connection records by the destination hosts, and applied the same pattern mining and feature construction process. Rather than using a time window of 2 seconds, we now used a "connection" window of 100 connections, and constructed a mirror set of "host-based traffic" features as the (time-based) "traffic" features.

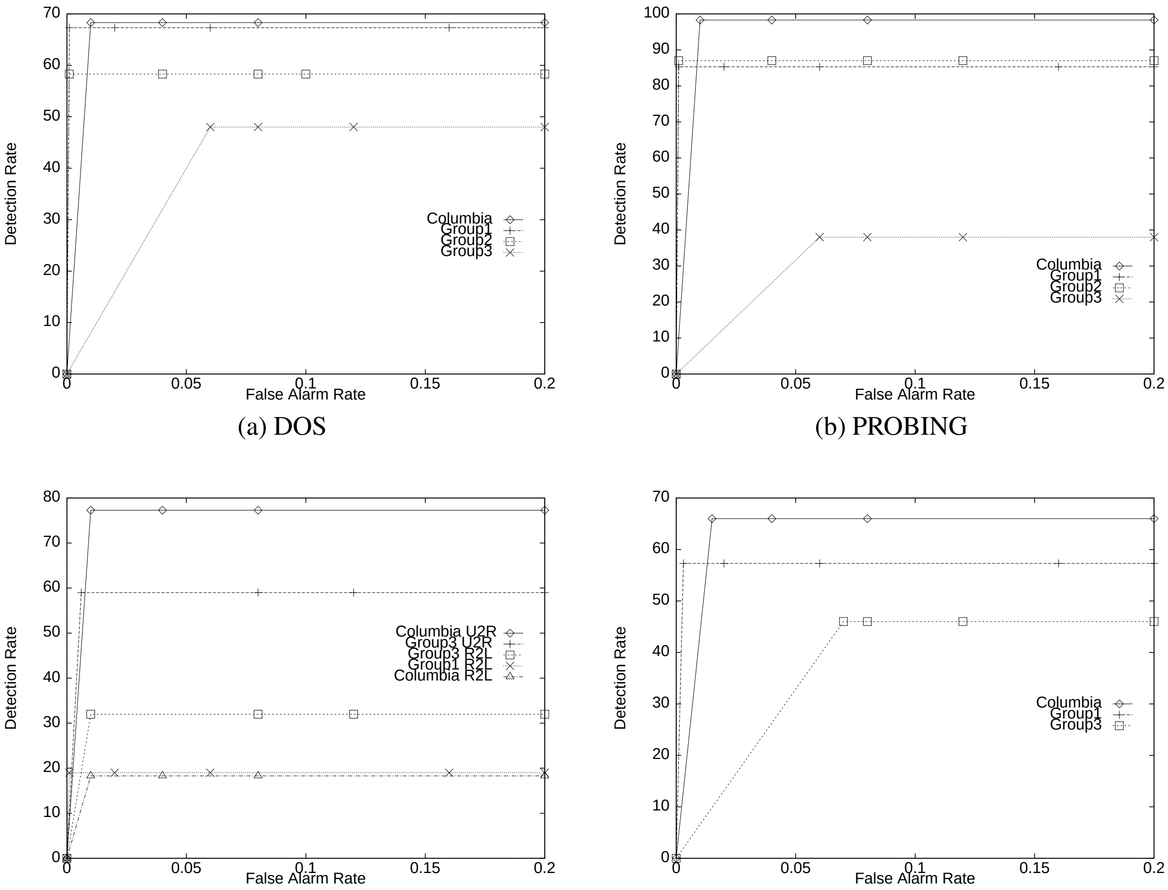
We discovered that unlike most of the DOS and PROBING attacks, the R2L and U2R attacks don't have any "intrusion only" *frequent* sequential patterns. This is because the DOS and PROBING attacks involve *many* connections to some host(s) in a very short period of time, the R2L and PROBING attacks are embedded in the data portions of the packets, and normally involves only a *single* connection. Algorithms for mining the unstructured data portions of packets are still under development. Presently, we use domain knowledge to add features that look for suspicious behavior in the data portion, e.g., number of failed login attempts. We call these features the "content" features.

Model	Feature set	Intrusion categories	# of features in records	# of rules	# of features used in rules
content	"intrinsic" + "content"	U2R, R2L	22	55	11
traffic	"intrinsic" + "traffic"	DOS, PROBING	20	26	4+**9**
host traffic	"intrinsic" + "host traffic"	Slow PROBING	14	8	1+**5**

Table 4: Model Complexities

We then built three specialized models, using RIPPER, that each has a different set of features and detects different categories of intrusions. For example, for the "content" model, each connection record contains the "intrinsic" features (see Section 3) and "content" features, and the resultant RIPPER rules detect U2R and R2L attacks. A meta-classifier was used to combine the predictions of the three base models when making a final prediction to a connection record. Table 4 summarizes these models. The numbers in bold, for example, **9**, indicate the number of automatically constructed temporal and statistical features being used in the RIPPER rules. We see that for both the "traffic" and host-based "traffic" models, our feature construction process contributes the majority of the features actually used in the rules.

We report here the performance of our detection models as evaluated by MIT Lincoln Labs. We trained our intrusion detection models, i.e., the base models and the meta-level classifier, using



(a) DOS (b) PROBING

(c) U2R and R2L (d) Overall

Figure 3: ROC Curves on Detection Rates and False Alarm Rates

the 7 weeks of labeled data, and used them to make predictions on the 2 weeks of unlabeled test data. The test data contains a total of 38 attack types, with 14 types in the test data only (i.e., our models were not trained with instances of these attack types).

Figure 3 shows the ROC curves of the detection models by attack category as well as on all intrusions. In each of these ROC plots, the x-axis is the false alarm rate, calculated as the percentage of normal connections classified as an intrusion; the y-axis is the detection rate, calculated as the percentage of intrusions detected (since the models produced binary outputs, the ROC curves are not continuous). We compare here our models with other participants (denoted as Group 1 to 3) in the DARPA evaluation program (these plots are duplicated from the presentation slides of a report given by Lincoln Labs in a DARPA PI meeting. The slides can be viewed on line via http://www.cs.columbia.edu/˜sal/JAM/PROJECT/MIT/mit-index.html). These groups primarily used knowledge engineering approaches to build their intrusion detection systems. We can see from the figure that our detection model have the best overall performance, and in all but one attack category, our model is one of the best two.

We discussed the Evaluation results with some researchers of other participating groups. It is agreed that the (manual) knowledge engineering approach suffers from the difficulties of dealing with large amount of data, and the inabilities to generalize the (often too-specific) hand-coded models. Our procedures for automatic construction features from mined patterns worked well with the large dataset. Using inductive classification rules also provides better performance on the "new" attacks. However, as the results on R2L shows, all models performed poorly when there are very large varieties of attack methods (of the same intrusion category). Much research is still needed in network intrusion detection.

4 Efficient Execution of Learned Rules

Our intrusion detection models are produced off-line. Effective intrusion detection should be in real-time to minimize security compromises. We therefore need to study how our models preform in a real-time environment. We are working on translating RIPPER rules into real-time detection modules in NFR (Network Flight Recorder) [NFR], a system that includes a packet capturing engine and N-code programming support for specifying packet "filtering" logic.

In our first implementation, we essentially tried to follow the off-line analysis steps in a real-time environment. A connection is not inspected (classified using the rules) until its connection record is completely formulated, that is, all packets of the connection have arrived and summarized, and all the temporal and statistical features are computed. This scheme failed miserably. When there is a large volume of network traffic, the amount of time taken to process the connection records within the past 2 seconds and calculate the statistics is also very large. Many connections may have terminated (and thus completed with attack actions) when the current connection is finally inspected by the RIPPER rules. That is, the detection of intrusions is severely delayed. Ironically, DOS attacks, which typically generate a large amount a traffic in a very short period time, are often used by intruders to first overload an IDS, and use the detection delay as a window of opportunity to quickly perform their malicious intent. For example, they can seize control of the operating system and "kill" the IDS.

We need to examine the time delay associated with each feature in order to speed up the model execution. In a "data flow" environment such as real-time intrusion detection, the time delay of a feature includes not only the time of its computation, but also the time of its readiness (i.e., when it can be computed). For example, the *flag* of a connection can only be computed (summarized) after the last packet of the connection has arrived, whereas the *service* of a connection can be obtained by checking the header of the first packet.

We partition the features into 3 "cost" (time delay) levels: level 1 features can be computed from the first packet; level 2 features can be computed at the end of the connection, using only information of the current connection; level 3 can be computed at the end of the connection, but require access to data of (many) other prior connections. As a datum arrives early in the "data flow", shown in Figure 1, the cost will be lower to calculate the feature that depends upon that datum. In order to conveniently estimate the cost of a rule, we assign a cost of 1 to the level 1 features , 10 to level 2, and 100 to level 3. That is, the different levels have an order of magnitude difference in cost. For the feature set derived from the DARPA dataset, *service* is a level 1 feature, all the other "intrinsic" and "content" features are in level 2, and all "traffic" features are in level 3.

Note that we cannot simply order the rules by their costs for real-time execution for the following reasons. First, the rules output by RIPPER are in a strict sequential order (e.g., "if rule 1 else rule 2 else ..."), and hence reordering the rules may result in unintended classification errors. Furthermore, even if the rules can be tested in strictly cost order without introducing classification error, many rules will still be tested (and fail to match) before a classification is made. That is, ordering the rules by their costs alone is not necessarily the optimal solution for quick model evaluation. We thus seek to compute an "optimal schedule" for feature computation, and rule testing to minimize model evaluation costs, and increase the response rate for real-time detection.

4.1 Low Cost "Necessary" Conditions

Ideally, we can have a few tests involving the low cost (i.e., level 1 and level 2) features to eliminate the majority of the rules that need to be checked, and thus eliminating the needs to compute some of the high cost features.

In order to eliminate a rule for intrusion I, we need a test of the form of $F \rightarrow \neg I$, which can be derived from $I \rightarrow \neg F$. We can then compute the association rules that have the intrusion labels on the LHS and the low cost features on the RHS, and a $confidence$ of 100%.

We discovered several such associations for the RIPPER rules, for example, $ping\_of\_death \rightarrow service = icmp\_echo$ [$c = 1.0$], $phf \rightarrow service = http$ [$c = 1.0$], and $port\_scan \rightarrow src\_bytes = 0$ [$c = 1.0$], $syn\_flood \rightarrow flag = S0$ [$c = 1.0$], etc. Note that most of these feature values, for example, $src\_bytes = 0$, are not in the RIPPER rules because they are prevalent in the normal data. That is, they don't have predictive power. However, these associations are the "necessary" conditions for the intrusions, for example, "this connection is a port-scan attack **only if** src_bytes is 0", which is equivalent to "if the src_bytes is **not** 0, then this connection is **not** a port-scan attack".

Note that when the RHS of such associations has n feature value pairs (regarding to different features), there are a corresponding n independent necessary conditions. We can always select the one with the lowest cost. We can also merge associations, $I \rightarrow A_i = v_1$ [c_1], $I \rightarrow A_i = v_2$ [c_2], $\ldots$, and $I \rightarrow A_i = v_n$ [c_n], where $\sum_{i=1}^{n} c_i = 1.0$, into a single association, $I \rightarrow A_i = v_1 \vee v_2 \ldots \vee v_n$ [$c = 1.0$]. For example, we have from the DARPA data, $buffer\_flow \rightarrow service = telnet$ [$c = 0.91$] and $buffer\_flow \rightarrow service = rlogin$ [$c = 0.09$], which are merged to $buffer\_flow \rightarrow service = telnet \vee rlogin$ [$c = 1.0$].

When a RIPPER rule for an intrusion is excluded because of the failure of its necessary condition, the features of the rule need not be computed, unless they are needed for other candidate (remaining) rules. We next discuss how to do efficient bookkeeping on the candidate rules and features to determine a schedule for feature computation and rule condition testing.

4.2 Real-time Rule Filtering

Suppose that we have n RIPPER rules. We use a n-bit vector, with the bit order corresponding to the order of the rules output by RIPPER, as the "remaining" vector to indicate which rules still need to be checked. Initially, all bits are 1's. Each rule has a "invalidating" n-bit vector, where only the bit corresponding to the rule is 0 and all other bits are 1's. Each of the high cost features, i.e., the level 3 temporal and statistical feature, has a "computing" n-bit vector, where only the bits corresponding to the rules that require this feature are 1's.

For each intrusion label, we record its "lowest cost necessary condition" (if there are such conditions), according to the costs of the features involved. We sort all these necessary conditions according to the costs to produce the order for real-time condition checking.

When examining a packet, or a (just completed) connection, if a "necessary" condition of an intrusion is violated, the corresponding "invalidating" bit vectors of the RIPPER rules of the intrusion are used to AND the "remaining" vector and all the "computing" vectors for the high cost features. After all the necessary conditions are checked, we get all the features with non-zero "remaining" vectors. These features are potentially useful because of the remaining rules that need to be checked. A single function call is made to N-code modules to compute all these features at once. This execution strategy reduces memory or disk access since these features compute statistical information on the past (stored) connections records. The "remaining" vector is then used to check the remaining rules one by one.

We are currently fine tuning our implementation of this scheme and need to perform an extensive set of experiments, simulating a wide variety of intrusions, to establish the empirical speed-up we may attain. (We will report our results when possible.) However, our analysis on the necessary conditions for DOS and PROBING attacks, and the set of features used by their RIPPER rules, suggest that one or two simple low cost tests (e.g., *service* and/or *flag*) can reduce the number of high cost feature tests from 9 (see Table 4) to at most 3. Our preliminary experiments have thus far confirmed this result.

5 Related Work

Our feature construction approach is similar to the work in [JH94]. Our "operators" are also based on extracted patterns of existing sets of features, and we as well consider the syntactic form of the patterns. We can use fewer and simpler operators, however, since the patterns carry "stronger" information (e.g., the invariant behavior of an intrusion). We also use automatic pattern encoding and comparison algorithms to produce input patterns for the feature construction program.

Our work is related to cost-sensitive learning, for example [Tur95], where both the cost of tests (and features) and accuracy are crucial criteria when building models. The cost of model evaluation adds a significant twist to this approach. We plan further study and comparison of these approaches.

In DC-1 (Detector Constructor) [FP97], a sequence of operations for constructing features (indicators) is needed before a cellular phone fraud detector, a classifier, is constructed. We have a harder problem here because there is no standard record format for connection records (unlike the phone call record which already provides rich structure for an individual customer phone account, we had to invent our own). We also need to construct temporal and statistical features not just for "individual accounts", but also over different connections, and services. That is, we are modeling different logical entities that take on different roles and whose behavior is recorded in great detail. Extracting these from a fast and overwhelming stream of packet data adds considerable complexity to the problem.

6 Conclusions and Future Work

In this paper we described the challenges as well as opportunities for KDD in a "data flow" environment. Using network intrusion detection as a concrete case study, we showed that the "expert-intensive" feature construction process (which is part of data preprocessing) can be guided and supported by data mining programs. Our approach is to encode and compare the frequent patterns mined from the normal and intrusion datasets, and automatically construct statistical and temporal features that describe the anatomy and invariant behavior of the attacks. The results from the DARPA evaluation show that the intrusion detection models produced using our method outperformed other other "knowledge-engineered" systems. We also showed that it is critical to consider the cost (time delay) of real-time execution of a model. We devised a simple scheme that aims to use a few low cost tests to filter out a large portion of the high cost feature computations.

As for future work, we will continue our research on optimizing learned rules for real-time execution. We will also study the issues of network anomaly detection, which is the only way possible to detect new attacks that are completely different in nature than any of the known intrusions.

7 Acknowledgment

We would like to thank Forster Provost of Bell Atlantic, and William Cohen of AT&T for helpful discussion.

References

[AIS93] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 207–216, 1993.

[Coh95] W. W. Cohen. Fast effective rule induction. In *Machine Learning: the 12th International Conference*, Lake Taho, CA, 1995. Morgan Kaufmann.

[CS93] P. K. Chan and S. J. Stolfo. Toward parallel and distributed learning by meta-learning. In *AAAI Workshop in Knowledge Discovery in Databases*, pages 227–240, 1993.

[FP97] T. Fawcett and F. Provost. Adaptive fraud detection. *Data Mining and Knowledge Discovery*, 1:291–316, 1997.

[FPSS96] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. The KDD process of extracting useful knowledge from volumes of data. *Communications of the ACM*, 39(11):27–34, November 1996.

[JH94] N. Japkowicz and H. Hirsh. Towards a bootstrapping approach to constructive induction. In *Working Notes of the Workshop on Constructive Induction and Change of Representation*, 1994.

[JLM89] V. Jacobson, C. Leres, and S. McCanne. tcpdump. available via anonymous ftp to ftp.ee.lbl.gov, June 1989.

[LSM98] W. Lee, S. J. Stolfo, and K. W. Mok. Mining audit data to build intrusion detection models. In *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining*, New York, NY, August 1998. AAAI Press.

[LSM99] W. Lee, S. J. Stolfo, and K. W. Mok. A data mining framework for building intrusion detection models. In *Proceedings of the 1999 IEEE Symposium on Security and Privacy*, May 1999.

[MTV95] H. Mannila, H. Toivonen, and A. I. Verkamo. Discovering frequent episodes in sequences. In *Proceedings of the 1st International Conference on Knowledge Discovery in Databases and Data Mining*, Montreal, Canada, August 1995.

[NFR] Network Flight Recorder Inc. Network flight recorder. http://www.nfr.com, 1997.

[Tur95] P. D. Turney. Cost-sensitive classification: Empirical evaluation of a hybrid genetic decision tree induction algorithm. *Journal of Artificial Intelligence Research*, 2(1995):369–409, 1995.