Collaborative Alert Ranking for Anomaly Detection

Ying Lin*,†
University of Houston
ylin53@central.uh.edu

Zhengzhang Chen*,#
NEC Laboratories America
zchen@nec-labs.com

Cheng Cao
Amazon Inc.
chengcao@amazon.com

Lu-An Tang
NEC Laboratories America
ltang@nec-labs.com

Kai Zhang
Temple University
zhang.kai@temple.edu

Wei Cheng
NEC Laboratories America
weicheng@nec-labs.com

Zhichun Li
NEC Laboratories America
zhichun@nec-labs.com

ABSTRACT

Given a large number of low-quality heterogeneous categorical alerts collected from an anomaly detection system, how to characterize the complex relationships between different alerts and deliver trustworthy rankings to end users? While existing techniques focus on either mining alert patterns or filtering out false positive alerts, it can be more advantageous to consider the two perspectives simultaneously in order to improve detection accuracy and better understand abnormal system behaviors. In this paper, we propose *CAR*, a *c*ollaborative *a*lert *r*anking framework that exploits both temporal and content correlations from heterogeneous categorical alerts. *CAR* first builds a hierarchical Bayesian model to capture both short-term and long-term dependencies in each alert sequence. Then, an entity embedding-based model is proposed to learn the content correlations between alerts via their heterogeneous categorical attributes. Finally, by incorporating both temporal and content dependencies into a unified optimization framework, *CAR* ranks both alerts and their corresponding alert patterns. Our experiments — using both synthetic and real-world enterprise security alert data — show that *CAR* can accurately identify true positive alerts and successfully reconstruct the attack scenarios at the same time.

CCS CONCEPTS

• **Information systems** → **Data mining**; *Enterprise information systems*; • **Security and privacy** → **Intrusion detection systems**; • **Computing methodologies** → **Anomaly detection**; *Temporal reasoning*; *Dimensionality reduction and manifold learning*; **Unsupervised learning**;

KEYWORDS

Anomaly detection; Alert ranking; Temporal dependency modeling; Content dependency modeling; Entity embedding; Enterprise security system

ACM Reference Format:
Ying Lin, Zhengzhang Chen, Cheng Cao, Lu-An Tang, Kai Zhang, Wei Cheng, and Zhichun Li. 2018. Collaborative Alert Ranking for Anomaly Detection. In *The 27th ACM International Conference on Information and Knowledge Management (CIKM'18), October 22–26, 2018, Torino, Italy.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3269206.3272013

1 INTRODUCTION

In modern information systems or cyber-physical systems — such as enterprise networks, OT/IoT systems, and data centers — a significant challenge is to understand the behavior of the system based on the massive data collected [6, 25]. By identifying underlying anomalies or irregular patterns [6, 7], critical actionable information can be extracted to facilitate human decisions and mitigate potential damage. Towards this end, a variety of anomaly detection engines (*e.g.*, [4, 20, 22]) have been developed to provide in-depth protections in various information systems.

Due to the overwhelming scale and complexity of real-world systems, however, these automated detection engines are notorious for generating high rates of false alarm [2, 30]. According to a recent study conducted by FireEye, most organizations receive 17,000 alerts per week and more than 51% of them are false positives and only 4% of the alerts are actually get investigated [13]. Due to the enormous amount of alerts, important alerts get lost easily in the noise of unimportant alerts and system analyst/administrators face "alert fatigue". This not only poses significant challenges to end-users in performing effective analysis and initiating the timely response, but can also suffer from prevalent false positive alerts, leading to unnecessary system interventions or suspensions. Therefore, it is particularly important to build trustworthy post-processing systems to reduce false positives from massive raw alerts, intelligently correlate different alerts, and uncover interpretable alert patterns for a better understanding of system irregularities.

Building an efficient alert post-processing system can be quite challenging for several reasons. First, most anomaly detection engines focus on low-level interaction/events (*e.g.*, a process accesses a sensitive file) and generate isolated alerts, while a system's abnormal behaviors are typically high-level activities composed of multiple low-level events/steps [12]. For example, in an enterprise security system, a well-known network attack called *Advanced*

*Both authors contributed equally.
†Work done during an internship at NEC Laboratories America.
#Corresponding author.

CIKM'18, October 22–26, 2018, Torino, Italy

ACM ISBN 978-1-4503-6014-2/18/10…$15.00
https://doi.org/10.1145/3269206.3272013

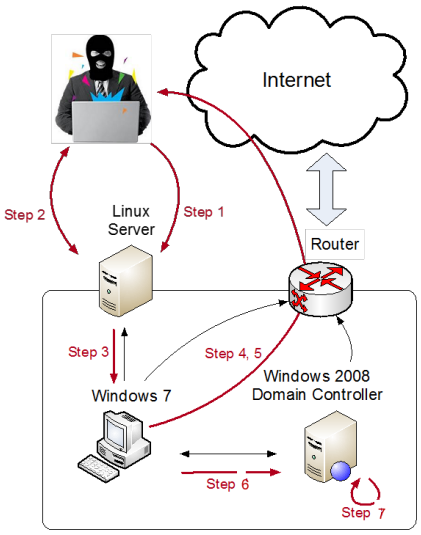


Figure 1: An example of APT attack

Persistent Threat (APT), as shown in Fig. 1, includes a sequence of computer hacking processes. Usually, the first attempt is to gain a foothold in the environment. Then, it uses the compromised systems as access to the target network, which is followed by deploying additional tools to fulfill the attack. Thus, it can be difficult to identify truly relevant processes/alerts to the attack by looking at them independently. Second, the multi-modal and nonlinear correlations between alerts further exacerbate the problem. Alerts can demonstrate complex inter-relations either in the temporal domain (temporal correlation) or in terms of their contents (content correlation). These contents are often represented by categorical attributes [37] that defy existing similarity measurements. Meanwhile, temporally correlated alerts may not occur consecutively in time but can be separated by false positive ones. Therefore, it is necessary to develop post-processing systems that capture both long-term temporal dependencies as well as content-level correlations.

Existing work on alert post-processing focuses either on modeling causalities between alerts to generate abnormal patterns [21, 36], or filtering out false positive alerts [28, 29]. The models on alert pattern discovery rely on the prior knowledge or alert labels, which is inadequate to be applied to real system when prior knowledge is not available and the proportion of false positive alerts is large. The models on false positive alert filtering, by contrast, ignore the correlations between alerts and deal with them independently. Therefore, it can be more advantageous to consider both objectives together to improve detection accuracy and understand complex abnormal behaviors. On the one hand, filtering out false positive alerts leads to more confident abnormal scenarios/patterns being reconstructed; on the other hand, modeling the dependencies between alerts helps filter out some isolated alerts as false positives.

In this paper, we propose *CAR*, an unsupervised, data-driven *c*ollaborative *a*lert *r*anking framework that enables one to identify true positive alerts and their corresponding alert patterns simultaneously, by exploiting both temporal and content correlations between alerts. *CAR* addresses the aforementioned challenges in three steps. To model the temporal dependencies between alerts, we design a prefix tree-based model that compactly represents alert sequences as well as preserves long-term temporal dependencies. A set of hierarchical Pitman-Yor priors are used to model the dependencies in a probabilistic way. We further discover a set of alert patterns that correlate individual alerts with higher-level abnormal behaviors. To model the content correlations between alerts, we propose an entity embedding-based model to learn the latent vector of each entity. Based on the latent representation, a proximity measurement is adopted to quantify the similarity between alerts. Finally, in order to simultaneously consider the temporal and context correlations, we propose the collaborative alert ranking via a unified optimization framework, such that low-level alerts and high-level alert patterns that truly relate to abnormal behaviors can be uncovered. Empirical studies on both synthetic and real enterprise alert data demonstrate the effectiveness of our proposed model.

To summarize, in this work we make the following contributions:

- We identify an important problem (collaborative alert ranking) in alert post-processing;
- We propose an optimization framework for collaborative alert ranking by exploiting both temporal and content correlations between alerts;
- We develop a prefix tree-based temporal model to discover the underlying abnormal scenarios by modeling the long-term dependencies between alerts;
- We build an embedding-based content model to measure the similarity between heterogeneous categorical alerts;
- We successfully apply our methodology to a real enterprise security system and demonstrate its effectiveness.

In the rest of this paper, we formally define our problem in Section 2, and provide the detail description of the proposed approach in Section 3. Experimental results of applying the proposed method in a real enterprise security system are presented in Section 4. Finally, we discuss the related work and draw the conclusion in Section 5 and Section 6, respectively.

2 PRELIMINARIES AND PROBLEM DEFINITION

Different from existing methods that focus on numerical data, our goal is to build an alert post-processing system for categorical alert data.

Heterogeneous Categorical Alert. A heterogeneous categorical alert $a = (v_1, \ldots, v_m, t)$ is a suspicious event record that contains m different categorical attributes, and the i^{th} attribute value v_i denotes an entity from the type V_i. The timestamp t can also be categorized into meaningful intervals (such as hours). For example, in enterprise security systems, an alert typically involves entities such as user, time, source/destination process, and files.

Alert Pattern. An alert pattern is a subsequence of alerts that may represent multiple steps of an abnormal system activity. An alert pattern of length L is constructed by L alerts, denoted as $u_{1:L} = \{a_1, \ldots, a_L\}$, ordered by their time stamps, $T(a_1) < \cdots < T(a_L)$. For instance, an ATP attack shown in Figure 1 can be deemed to be

an alert pattern, consisting of a number of steps/alerts in it. Each alert indicates one step in the attack.

Problem Statement: Collaborative Alert Ranking. Given a set of categorical alerts $\{a_1, \ldots, a_T\}$ generated by an anomaly detection engine, the maximum length of an alert pattern L_{max}, and an integer number K, the problem is to identify top K ranked alerts that are most likely to be true positives and their corresponding alert patterns with a length no more than L_{max}.

3 METHODOLOGY

3.1 Overview

The intuition behind our method is that the alerts whose occurrence can be collectively related and supported by each other, both temporally and contextually, are more likely to shed important lights on true abnormal events. Thus, our goal is to automatically explore temporal and content-based dependencies between alerts to rank the potential true positive alerts and their sequential patterns higher. In some rare cases, the isolated alerts may be true positives. But it's possible to identify these isolated true positive alerts by using some empirical evidence (*e.g.*, via a reference), which is beyond the scope of this paper (*i.e.*, designing a data-driven method).

We start by exploiting the temporal structure in alert sequences. A prefix tree-based hierarchical Bayesian model is built to recover temporal dependencies between alerts and simultaneously extract alert patterns. In the meantime, we embed the categorical entities/attributes in a latent space to model content similarities between alerts. Finally, we compute the ranking of alerts by maximizing the consensus among the temporal and content structures, and simultaneously generate the alert patterns from the true positive alerts.

3.2 Temporal Dependency Modeling

Since a real system's abnormal behavior, such as the APT attack, may include a number of separated steps residing on a long temporal span, we aim at capturing the dependencies between nearby alerts (short-term dependency) as well as alerts in long patterns (long-term dependency).

Intuitively, for each possible alert pattern $u_{1:L} = \{a_1, \ldots, a_L\}$ from the alert sequence, we can measure the temporal dependency between each alert $a_t, t \geqslant 2$ and its preceding alerts $u_{1:t-1} = \{a_1 \ldots, a_{t-1}\}$ using a predictive distribution conditioned on $u_{1:t-1}$, *i.e.*, $p(a_t|u_{1:t-1})$. The overall strength of temporal dependency in each alert pattern can be assessed by the joint distribution, $p(u_{1:L})$. A higher joint probability indicates a stronger temporal dependency in the alert pattern. Using the Bayesian rule, the joint distribution of an alert pattern can be estimated by a sequence of predictive distributions:

$$p(u_{1:L}) = \prod_{t=1}^{L} p(a_t|u_{1:t-1}). \tag{1}$$

Here, we assume the distributions are discrete and the number of possible alerts is finite. But every alert would be unique when all attributes, including time-related attributes, are considered, and we would lose the opportunity to learn any temporal patterns. Therefore, we consider a set of important entities with examples including the alert event source (*e.g.*, process exename) and destination (*e.g.*, file name) to represent each alert[1] and symbolize the alert sequences with a finite symbol set Σ. That is, the alerts with the same source and destination will be represented using the same symbol.

Denote the predictive distribution conditioned on preceding alerts as $G_{[u_{1:t-1}]}$. To obtain the joint distribution in Eq. 1, we need to estimate all predictive distributions conditioned on the short pattern included in u, *i.e.*, $\{G_{[u']}\}_{u' \in \{u_{1:t}|1 \leqslant t \leqslant L\}}$. However, estimating each predictive distribution independently requires adequate training observations that include the specific alert pattern, which often can not be satisfied in real-world scenarios.

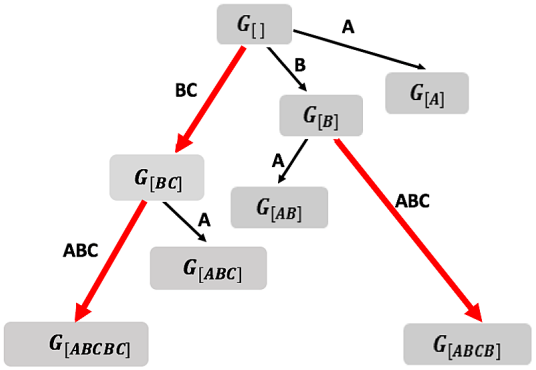


Figure 2: An example of prefix tree on sequence *ABCBC*. The marginalized edges are marked in bold red.

3.2.1 Prefix Tree Construction. To solve this problem, we first use a prefix tree to efficiently represent the alert sequence and hierarchically tie together the predictive distributions. The prefix tree preserves the temporal relationships between alerts [34]. Specifically, each node in the prefix tree represents an alert pattern and the root is associated with empty pattern. The descendants of an alert pattern represent its prefix patterns that include it and add a prefix alert to it. By recursively adding prefix alerts to the alert pattern and marginalizing out the non-brunching interior nodes, the prefix tree can be directly constructed from an input sequence in linear time and space [34]. An example sequence $\mathbf{s} = ABCBC$ is used in Fig. 2 to illustrate the construction of prefix tree.

3.2.2 Hierarchical Bayesian Model Learning. Then, we build a hierarchical Bayesian model to estimate the predictive distributions connected by the prefix tree. Each pattern in the prefix tree is associated with a predictive distribution conditioned on it and the descendants of it represent the predictive distributions of its prefix patterns. Our model uses observations that occur in each pattern to recursively inform the predictive distributions of its prefix patterns and vice versa.

To model the discrete predictive distributions, one typical way is by using Dirichlet priors. However, recent research [9] has shown

[1]Please note that other entity attributes will still be considered in the content dependency modeling part (see Section 3.3).

that models employing Pitman-Yor priors can significantly outperform the Dirichlet priors, especially where complex hierarchical relationships exist between latent variables. Thus, in this work, we apply a set of hierarchical Pitman-Yor processes to model the discrete predictive distributions in the prefix tree and their hierarchical structure. We model each predictive distribution, $G_{[u]}$, as a Pitman-Yor process, $PY(d, c, G_0)$, whose random sample comes from an infinite discrete probability distribution. It is governed by a prior distribution, G_0, representing the prior probability of occurrence of each alert before observing any data; a discount parameter d ($0 \leqslant d \leqslant 1$), and a strength parameter c ($c < -d$), controlling the amount of variability around G_0. When $d = 0$, the Pitman-Yor process reduces to a Dirichlet distribution with parameter cG_0. Using the Pitman-Yor process, each predictive distribution can be expressed as:

$$G_{[u]}|d_u, c_u, G_{[\pi(u)]} \sim PY(d_u, c_u, G_{[\pi(u)]}), \tag{2}$$

where d_u and c_u are the discount and strength parameters and $G_{[\pi(u)]}$ is the prior distribution of $G_{[u]}$. The prior distribution corresponds to the predictive distribution conditioned on its suffix pattern, $\pi(u)$, which consists of all but the earliest alert in u. It's also a Pitman-Yor process. This choice of the prior structure reflects our belief that among all preceding alerts, those appearing closer to the current alert are more important in capturing the dependencies with the current alert. All parameters can be computed using Gibbs sampling.

3.2.3 Alert Pattern Searching. We are interested in finding the alert patterns corresponding to abnormal behaviors by their temporal dependencies. In practice, it can be time-consuming to search all possible alert patterns in a sequence, in particular considering long-term dependencies. To address this problem, we use the breadth-first search to traverse the prefix tree to identify the temporally dependent alert patterns, and use the joint distribution in Eq. 1 to measure the dependency in each alert pattern. The stronger the temporal dependency in a pattern is, the more likely that alerts in this pattern collectively pinpoint an abnormal behavior. Since the length of the pattern can grow very large as the height of prefix tree increases, we use parameters L_{max} and L_{min} to control the maximal and minimal lengths of patterns. Therefore, we obtain a set of alert patterns and their strengths of dependencies from the temporal model, denoted as $U = \{(u, p_u)|L_{min} \leqslant |u| \leqslant L_{max}\}$. For example, if the L_{max} and L_{min} are set at 3 and 2, respectively, three patterns, $\{BC, ABC, AB\}$ can be obtained from the prefix tree shown in Fig. 2.

3.3 Content Dependency Modeling

In addition to temporal information, an alert also contains rich content information in terms of heterogeneous categorical entities such as host, user, source, and destination. The content similarities among a series of alerts can provide useful hints to group them into the same high-level anomalies. In this section, we explore the content dependencies of these categorical entities within each alert.

Every alert consists of a set of entities. Their co-occurrence in different alerts is one of the most commonly considered connections [1]. However, entities — as the categorical attributes of an alert — don't have any intrinsic ordering among themselves, making traditional numerical learning algorithms hard to directly leverage their co-occurrences [37]. To address this challenge, we embed all entities into a common latent space, where co-occurrences between entities are preserved. Then, the closeness between entities can be easily calculated in the space by popular metrics, such as cosine similarity.

In concrete, given an alert $a = (v_1, v_2, \ldots, v_m, t)$, we model the entity co-occurrences in a by the conditional probability $p(v_i|v_j; \theta)$ for each pair of categorical entities in a, using the following Softmax function:

$$\prod_{v_i, v_j \in a} p(v_i|v_j; \theta) = \prod_{v_i, v_j \in a} \frac{\exp(w_{v_i v_j} \cdot z_{v_i} \cdot z_{v_j})}{\sum_{v_i' \in V} \exp(w_{v_i v_j} \cdot z_{v_i'} \cdot z_{v_j})}, \tag{3}$$

where z_{v_i} and z_{v_i} are the embedding vectors for v_i and v_j, respectively, and V is the set of all available entities. $w_{v_i v_j}$ is the weight for pairwise interaction between v_i's and v_j's entity types, and it is non-negative constrained. θ includes the parameters z_{v_i} and $w_{v_i v_j}$ for each $v_i \in V$.

Given all observations $\{a_1, \ldots, a_T\}$, our goal is to set the parameters such that Eq. 3 is maximized. This optimization model has been historically known to be very expensive, due to the denominator of Eq. 3 summing over all entities of V. Therefore, we follow the idea of *negative sampling* [26] to address this challenge. Negative sampling shares a very similar idea of *noise-contrastive estimation (NCE)* [15]. In general, to avoid dealing with too many entities in V, we only update a sample of them. We surely should keep all observed co-occurred entities in our data, and we need to artificially sample a few "noisy entity pairs" — they are not supposed to co-occur so their content similarities should be low. In the end, after taking the logarithm and negative sampling, we aim to minimize the following objective:

$$\begin{aligned} \min_{\mathbf{v}, \mathbf{w}} & - \textstyle\sum_{(v_i, v_j) \in D} \log(w_{v_i v_j} \sigma(z_{v_i} \cdot z_{v_j})) \\ & - \textstyle\sum_{(v_i', v_j') \in D'} \log(w_{v_i' v_j'} \sigma(-z_{v_i'} \cdot z_{v_j'})), \end{aligned} \tag{4}$$

where σ is the sigmoid function, and D is the collection of entity co-occurrences observed in our data. D' is the set of negative samples constructed by certain sampling scheme. Concretely, for each co-occurrence $(v_i, v_j) \in D$, we sample k noises $(v_{i_1}', v_j), (v_{i_2}', v_j), \ldots, (v_{i_k}', v_j)$, where v_i' is drawn according to a noise distribution. Without a rule of thumb on guiding negative sampling, we empirically test several and find the best one when sampling v_i in a probability inversely proportional to the frequency of co-occurrence with v_j. Then, the standard mini-batch gradient descent algorithm is used to solve the Eq. 4.

Next, given the learned embedding vector for each entity, we measure the pairwise alert similarity via the weighted combination of cosine similarities between their entities. Suppose we have two alerts $a_i = (v_{i_1}, v_{i_2}, \ldots, v_{i_m})$ and $a_j = (v_{j_1}, v_{j_2}, \ldots, v_{j_m})$. Their content similarity, denoted by S_{ij}, can be calculated as:

$$S_{ij} = \sum_{v_{i_k} \in a_i, v_{j_k} \in a_j} w_{v_{i_k}, v_{j_k}} \langle z_{v_{i_k}}, z_{v_{j_k}} \rangle,$$

where $z_{v_{i_k}}$ and $z_{v_{j_k}}$ are embedding vectors of entities v_{i_k} and v_{j_k}, respectively, and $w_{v_{i_k} v_{j_k}}$ is the learned weight between the two entity types of v_{i_k} and v_{j_k}.

3.4 Collaborative Alert Ranking

In this section, we propose a ranking algorithm that simultaneously computes the confidence scores for both individual alerts and alert patterns, by maximizing their consensus in terms of both temporal structure and content similarity.

Given a sequence of alerts a_i, $i = 1, \ldots, T$, whose confidence scores are denoted by τ_i's, $0 \leqslant \tau_i \leqslant 1$. The pairwise similarities between alerts is denoted by a similarity matrix $S \in \mathbb{R}^{T \times T}$. On the other hand, we also extract L alert patterns, u_l, $l = 1, \ldots, L$, based on the temporal model, whose strength of temporal dependency is denoted as p_l and provided by the temporal model. Note that for each alert pattern u_l, we have a number of alerts associated with it. The patterns constructed by true positive alerts are more likely to associate with the attack behavior. Thus, we also assign a confidence score to each alert pattern using μ_l's, $0 \leqslant \mu_l \leqslant 1$. The relation between T alerts and L alert patterns are represented by a 0/1 coincidence matrix $F \in \mathbb{R}^{T \times L}$. If alert a_i belongs to alert pattern u_l, then $F_{il} = 1$.

Our goal is to simultaneously compute the alerts' confidence τ_i's and alert-patterns' confidence μ_l's, given alerts' similarity S, coincidence matrix F, and temporal dependency strength within patterns, p_l's. Here, a key assumption is that alerts or alert patterns truly associated with abnormal system behaviors are more likely "supporting" each other in terms of either content correlations or temporal dependencies. Therefore, we intend to maximize the consensus between temporal and content dependencies following the requirements: 1) Those alert patterns with higher levels of temporal dependency strength, p_l, are more likely to be true positives; namely those alert patterns composed of temporally correlated alerts are more relevant patterns; 2) Alerts with similar content information tend to have similar confidence scores; 3) The confidence score of each alert pattern should be close to the confidence scores of the individual alerts associated with it.

Given these constraints, we propose the following optimization problem:

$$\begin{aligned} \min_{\tau,\mu} & -\sum_l p_l \mu_l + \frac{\lambda_1}{2} \sum_{i,j} S_{ij} (\tau_i - \tau_j)^2 \\ & + \frac{\lambda_2}{2} \sum_{i,l} F_{il} (\mu_l - \tau_i)^2 \\ s.t. & \ \sum_i \tau_i \leqslant K, \ 0 \leqslant \mu_l \leqslant 1, \ 0 \leqslant \tau_i \leqslant 1. \end{aligned} \tag{5}$$

Here, the first term in the objective function is used to correlate μ_l's to p_l's (requirement-1). The second term is used to enforce the smoothness of τ_i's based on the content-level similarity S (requirement-2). These two terms independently impose constraints on the alert-patterns' confidence and alerts' confidence. They are connected with each other by the third term in the objective, which states that the alert-pattern's confidence μ_l should be bounded close to the confidence of those individual alerts associated with the pattern $\tau_i, i \in \{j | a_j \in u_l\}$. By doing this, each alert and alert-pattern will be given a confidence score based on a global ranking that reaches the consensus as specified by the temporal structures and content similarities. λ_1 and λ_2 are tuning parameters controlling the effects of last two terms. A larger λ_1 encourages similar alerts to have similar confidence scores, and a larger λ_2 leads to closer confidence scores between alert-patterns and associated alerts. The confidence scores are non-negative, and K is a pre-defined integer that roughly controls the number of alerts with non-zero scores.

Proof of Convexity. Here, we prove the convexity of the optimization problem (Eq.5):

First, we write Eq. 5 in a matrix form. Define diagonal matrices $D_{F_c} \in \mathbb{R}^{L \times L}$ and $D_{F_r} \in \mathbb{R}^{T \times T}$, whose diagonal entries are $D_{F_c}(l,l) = \sum_i F_{il}$, $D_{F_r}(i,i) = \sum_l F_{il}$.

Use the Laplacian matrix L_S to represent similarity structure between alerts, *i.e.*, $L_S = D_S - S$, where D_S is a diagonal matrix with $D_S(i,i) = \sum_j S(i,j)$.

Let $\mathbf{x} = [\tau_1, \ldots, \tau_T, \mu_1, \ldots, \mu_L]^T \in \mathbb{R}^{(T+L)\times 1}$, then Eq. 5 can be simply written as:

$$\begin{aligned} & \min_{\mathbf{x}} \mathbf{q}^T \mathbf{x} + \tfrac{1}{2} \mathbf{x}^T \mathbf{Q} \mathbf{x} \\ & s.t. \ \mathbf{A}\mathbf{x} \leqslant \mathbf{b}, \ \ \mathbf{x} \geqslant \mathbf{0} \end{aligned} \tag{6}$$

where $\mathbf{q} = [\mathbf{0}, p_1, \ldots, p_L] \in \mathbb{R}^{1 \times (T+L)}$, $\mathbf{Q} = \begin{bmatrix} \lambda_2 D_{F_r} + \lambda_1 L_S & -\lambda_2 F \\ -\lambda_2 F^T & \lambda_2 D_{F_c} \end{bmatrix} \in \mathbb{R}^{(T+L)\times(T+L)}$, $\mathbf{A} = \begin{bmatrix} \mathbf{I}_{(T+L)\times(T+L)} \\ \mathbf{1}_{1\times T}, \mathbf{0}_{1\times L} \end{bmatrix} \in \mathbb{R}^{(T+L+1)\times(T+L)}$, and $\mathbf{b} = [1, \ldots, 1, K]^T \in \mathbb{R}^{(T+L+1)\times 1}$.

Since the Hessian matrix $\mathbf{Q}$ is positive semidefinite, *i.e.*, $\mathbf{x}^\mathbf{T}\mathbf{Q}\mathbf{x} = \lambda_1 \sum i, j S_{ij} (\tau_i - \tau_j)^2 + \lambda_2 \sum_{i,l} F_{il} (\mu_l - \tau_i)^2 \geqslant 0$ for $\forall \mathbf{x} \in \mathbb{R}^{(T+L)\times 1}$, the optimization problem in Eq. 6 is convex.

Thus, this is a standard quadratic programming problem that can be efficiently solved by existing algorithms such as NNLS [19]. By solving this problem, each alert and alert-pattern will be given a confidence score based on a global ranking that reaches the consensus as specified by the temporal structure and content similarity.

4 EXPERIMENTS

4.1 Experiment Setup

We apply *CAR* to a real-world enterprise intrusion detection system The alert data was generated by an off-the-shelf anomaly detection engine. The anomaly detection engine is able to automatically detect some unknown cyber-attacks, but meanwhile, it suffers from the same problem—high false positive rate—as many other anomaly-based intrusion detection engines. Hence, we apply *CAR* to help improve the intrusion detection performance. In total, there are 3,322 alerts collected from 173 computers within one month. Among them, 41 alerts are true positive ones corresponding to six different types of popular attacks. Each alert is a computer system event recording an interaction between a pair of system entities. The types of system entities include *processes*, *files*, and *Internet sockets* (INETSockets).

Attack Description. There are six different types of attacks consisted by 3 to 7 attack steps. Each step corresponds to one true positive alert and consecutive steps can be separated by false positives.

- *MLS Attack (MLS)*: This attack targets at the */selinux/mls* file, which defines the MLS (Multi-Level Security) classification of files within the host. In general, the */selinux/mls* file should be kept secret to all users except for the security administrator, as it exposes the security rules of a computer system and enables the attackers to find potential vulnerabilities. By the intrusion attack, the attacker first exploits the *ssh* process to access */selinux/mls* file. If the file access is successful, the file content is sent to an external host (*i.e.*, the attacker).

- *Snowden Attack (SNO)*: This attack targets at the */etc/passwd* file, which stores the password digest of all users as well as the user group information. First, the attacker tries to access the */etc/passwd* file by the *gvfs* process, which enables easy access from a remote host via FTP. Then the attacker tries to send the file via an *INETSocket*.
- *Botnet Attack (BOT)*: In this attack, the remote intruder employs the *bash* process to scan a sensitive file */var/log/apt/history.log*. This file stores detailed installation messages. The attackers are interested in it as they can intrude the host by exploiting the vulnerabilities of the installed software. The sensitive information is leaked via an *INETSocket*.
- *Emulating Enterprise Environment (EEE)*: This attack consists of seven steps. The hacker first utilizes the IRC vulnerability to create telnet process. The telnet process is then used to create malware process and open reverse connection. The malware process downloads malware binary (*trojan.exe*). Then the *trojan.exe* is created and used to connect back to the attacker host. DLL is injected by the running process *notepad.exe* and creates the connection back to the hacker. The hacker uses *mimikatz* and *kiwi* to perform memory operation inside the *meterpreter* context. Finally, malware *PwDump7.exe* and *wce.exe* are copied and run on the target host.
- *Diversifying Attack Vectors (DAV)*: There are six steps in this attack. The hacker first writes malicious PHP file by HTTP connection, then downloads the malware process (*trojan.exe*), and connects back to attacker host. The process *notepad.exe* is run to perform DLL injection. Attacker further uses *mimikatz* and *kiwi* to perform memory operation inside meterpreter context. Finally, it copies and runs *PwDump7.exe* and *wce.exe* on the target host.
- *Domain Controller Penetration (DCP)*: This attack includes five steps to penetrate domain controller. First, the hacker sends an email with a malicious word document, and the document writes malware *python32.exe*, which opens up a connection to the attacker host. The hacker then runs *notepad.exe* and performs reflective DLL injection to gain privilege. He/she further transfers password enumerator and runs process *gsecdump-v2b5.exe* to get all user credentials. Finally, SQL server address is probed to connect and dump the database into the attacker host.

Since real attacks typically have more than two steps to be accomplished, we set the minimum alert pattern length $L_{min} = 3$. And by default, we set the maximum alert pattern length $L_{max} = 7$, and the pre-defined integer $K = 50$.

Baseline Models. To demonstrate the effectiveness, we compare *CAR* with temporal-based methods (*i.e.*, *NGRAM* and *iBOAT*), content-based methods (*i.e.*, *Embedding* and *SimRank*), as well as the multi-view based method *MVCluster*.

- *NGRAM* [3]: This method has been widely used to learn the temporal structure between normal events, and label the alerts that do not appear in the normal patterns as abnormal ones. In our experiments, we use the first 40% false positive alerts for training.
- *iBOAT* [5]: This association rule based method defines true positive alerts as those whose corresponding temporal patterns have high confidence scores.
- *Embedding*: This is our content dependency modeling method (see Section 3.3) to learn the pairwise content-level similarities between alerts. We use values in the dominant eigenvectors of similarity matrices to determine the anomaly degree of each alert.
- *SimRank* [17]: *SimRank* is a widely-used graph-based model to measure the similarities between categorical data. It aggregates entities into a graph and measures the pairwise similarities by applying random walk algorithm. Based on the similarity matrix, we use the same way as in *Embedding* to measure the anomaly degree of each alert.
- *MVCluster* [31]: The multi-view based clustering method seeks groupings that are consistent across different representations of alert data. We use temporal and content attributes as views and cluster alerts into true positive and false positive groups.

Evaluation Metrics. Similar to [11, 12], we adopt ROC (Receiver Operating Characteristic curves) and PRC (Precision Recall curves) for evaluating the ranking scores. Both curves reflect the quality of predicted scores according to their true labels at different threshold levels. To get quantitative measurements, the AUC (Area Under curve) of both ROC and PRC are computed. For the method (*i.e.*, *MVCluster*) without predicted scores, we use precision, TPR (True Positive Rate), and FPR (False Positive Rate) instead.

Table 1: AUC of different alert ranking methods

Method	CAR	Embedding	SimRank	NGRAM	iBOAT
ROC	**0.998**	0.353	0.258	0.440	0.140
PRC	**0.719**	0.003	0.003	0.006	0.040

Table 2: Comparison between MVCluster and CAR

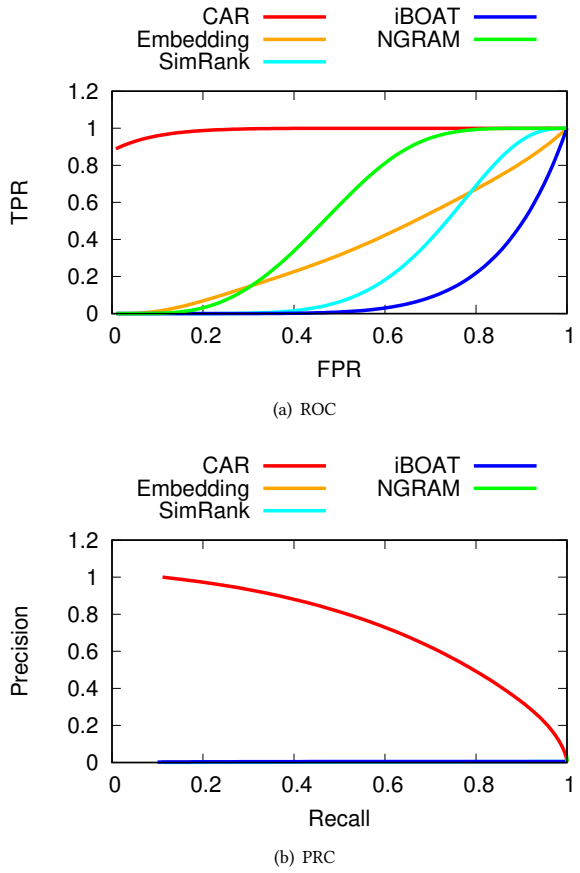
Method	TPR	FPR	Precision
MVCluster	1	0.720	0.006
CAR	1	**0.010**	**0.281**

4.2 Results for Detecting True Positive Alerts

The ROC and PRC curves presented in Fig. 3 demonstrate the effectiveness of the proposed method in detecting true positive alerts. As shown in the ROC curve, *CAR* is able to detect most true positive alerts when the false positive rate is controlled at a low level. The advantage of the proposed method is further quantified by the AUC values summarized in Table 1. In the PRC curve, due to the fact that a small number of true positives were ranked relatively low by the proposed method, the precision can be low if all true positives need to be detected. In addition, precision rates of the baseline methods are relatively low because these models can only filter out up to 50% false positive alerts in their best cases.

Table 3: AUC w.r.t. different parameter settings

Parameter	K = 50				L_{max}=7			
Value	L_{max}=4	L_{max}=5	L_{max}=6	L_{max}=7	K=1	K=10	K=50	K=100
ROC	0.995	**0.999**	0.998	0.998	0.996	0.998	0.998	**0.999**
PRC	0.401	0.829	**0.843**	0.840	0.760	0.760	0.802	**0.821**

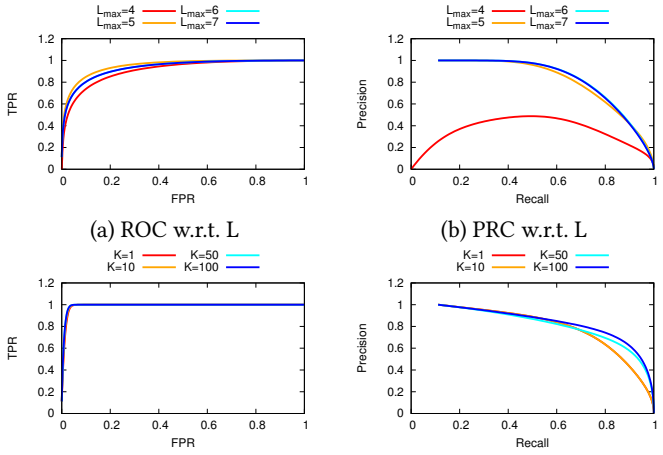


(a) ROC

(b) PRC

Figure 3: ROC and PRC curves of different methods.

In practice, it is unrealistic to provide accurate estimations of the length of alert patterns, neither the number of true positive alerts. Thus, we evaluate the effect of these parameters on the model performance. Since the patterns with the length greater than 7 are false positives with low confidence scores, they are unlikely to affect the ranking. We vary the maximum pattern length L_{max} from 4 to 7. We further set the pre-defined integer K at 1, 10, 50 and 100. The ROC/PRC curves w.r.t. L_{max} and K are plotted in Fig. 4. And the AUC values are summarized in Table 3. We observe that the pre-defined integer K doesn't affect the *CAR* performance, because it only controls the sparsity of alert scores but not the ranking among top alerts. Meanwhile, incorporating longer patterns will benefit the ranking result as demonstrated by the improvements of AUC when L_{max} increases. For instance, the PRC curve under $L_{max} = 4$ performs worse than the other scenarios and starts from (0, 0) point.



(a) ROC w.r.t. L

(b) PRC w.r.t. L

(c) ROC w.r.t. K

(d) PRC w.r.t. K

Figure 4: ROC and PRC curves w.r.t. the maximum pattern length L_{max} and the pre-defined integer K.

This is due to the fact that most attack scenarios have more than 4 attack steps and these long attack patterns are not considered when L_{max} is set at low level. This indicates the proposed algorithm is more effective when both short and long patterns are considered.

4.3 Attack Scenario Related Patterns

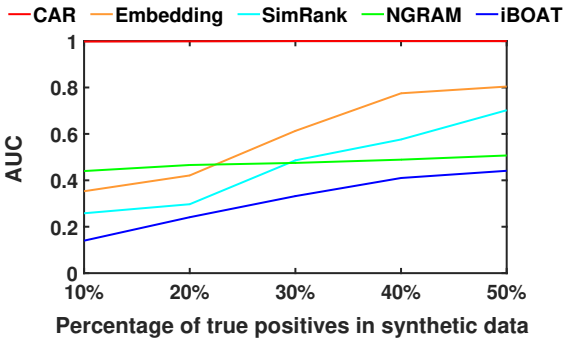
To evaluate the *CAR* method in terms of uncovering meaningful alert patterns, we compare it with the *Temporal* model presented in Section 3.2. Table 4 lists the top six alert patterns generated by *CAR* and compares their rankings in the *Temporal* model. These six patterns are all highly associated with the real attacks. We use the attack name plus the corresponding step number to represent the alert in each pattern. For instance, the alert *EEE-5* in the first pattern represents the fifth step of *Emulating Enterprise Environment* attack. Table 4 shows that by exploiting the content correlation between alerts, *CAR* greatly boosts the rank of attack-related patterns in *Temporal* model. For instance, four patterns are detected for the *EEE* attack. A longer pattern that captures the whole *EEE* attack scenario is also detected by *CAR*, with the highest rank in patterns with lengths larger than 4, but with a relatively low rank of 61 in all patterns. That is because longer patterns tend to have lower correlation scores.

4.4 Synthetic Experiment and Analysis

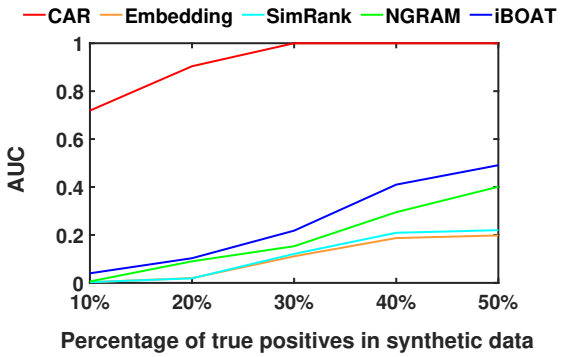
We further evaluate the robustness of *CAR* on five synthetic datasets that contain different percentages of true positive alerts (TPs). More specifically, the five synthetic datasets have 10%, 20%, 30%, 40% and

Table 4: High-ranked alert pattern related to attacks

CAR rank	Alert pattern	*Temporal* rank
1	EEE-5, EEE-6, EEE-7	1
2	EEE-1, EEE-2, EEE-3	2
3	EEE-4, EEE-5, EEE-6	3
4	DCP-2, DCP-3, DCP-4, DCP-5	7
5	DAV-2, DAV-3, DAV-4, DAV-6	14
6	EEE-1, EEE-2, EEE-3, EEE-4	41



(a) ROC



(b) PRC

Figure 5: AUC of different methods under five synthetic datasets.

50% TPs, respectively. They are generated by resampling the six attack scenarios and randomly injecting the resampled TPs to the 3,322 alerts generated from the real intrusion detection system. The temporal and content dependencies between TPs are preserved in the synthetic datasets.

We compare the detection accuracy of the proposed method *CAR* and baseline models. As shown in Fig. 5, the advantage of *CAR* is consistent under different percentages of true positives. The performance of content similarity based methods (*i.e., Embedding* and *SimRank*) is improved when the number of TPs increases. This is due to the fact that content dependency between TPs is strengthened by resampling. Since the temporal dependency in synthetic datasets is also stronger, the performance of *iBOAT* is also getting improved with the increased number of TPs. The *NGRAM* method, which uses a set of false positive alerts for training, is less likely to be affected by the percentage of TPs, with the AUC of the ROC remain unchanged.

5 RELATED WORK

5.1 Alert Post-processing

Data mining and machine learning techniques have been widely used in alert post-processing [18, 27]. Depending on the level of prior knowledge needed, they can be classified into supervised models, which rely on the labeled training alerts, and unsupervised models, which aim to detect unknown anomalies. Alert classification models [29] use classifiers to distinguish true alerts with false positives. These methods can not capture alerts that are not observed in the training phase. In order to compensate for the lack of prior knowledge, clustering techniques have been proven to be highly effective in reducing a large number of alerts by grouping similar events together [28]. Valdes and Skinner proposed a probabilistic model that measures the overall similarity between alerts as a weighted combination of their similarities on respective attributes [35]. However, the specification of weights relies on expert knowledge. Holmann and Sick proposed an online intrusion alerts aggregation system [16], which measures the similarity between alerts using a finite mixture distribution. However, the effectiveness of this method depends on the distribution assumptions. In addition, the main weakness of clustering models is that they do not exploit temporal dependencies, such as the causality between alerts.

5.2 Alert Pattern Discovery

Alert pattern discovery aims to reconstruct multi-step abnormal scenarios by looking into the causality between alerts. Association rule mining is used to investigate multi-step alerts reconstruction in [21]. Frequent sequence patterns are discovered over alert bursts and abnormal scenarios are constructed based on pattern matching and correlativity calculation. However, the risk of missing relevant alerts and the effects of alert latency are not considered in this work. Machine learning methods, such as n-gram analysis [10], Hidden Markov Model [36], and Bayesian network [27], are commonly used to reconstruct complex scenarios by training the model under known abnormal scenarios or true positive alerts. However, the effectiveness of these methods rely on the prior knowledge to acquire labeled data.

5.3 Multi-view Unsupervised Learning

Our work is also closely related to the multi-view unsupervised learning problems which aim to learn a consensus pattern from multiple representations of data[8, 24, 32, 33]. In social network analysis, for instance, same instances could be related to each other in multiple representations including email networks, collaboration networks, and organization hierarchy [24]. [24] proposed general optimization models for clustering and spectral dimensionality reduction from multi-view data while [33] proposed a multi-view feature selection framework to exploit relations among different views to help each other select relevant features. [14, 23] further used the multiple sources data for detecting anomalies by exploring

the inconsistent behaviors across different sources. However, the multi-view unsupervised learning methods only focus on exploiting the consistencies shared by graphical or vector representations from different sources. Without explicitly modeling the temporal and content correlations between instances, the multi-view learning methods are not able to be applied to this problem.

Different from the existing methods, *CAR* exhibits several desirable properties required by an alert post-processing system: it is (1) unsupervised: no training labels need to be provided; (2) data-driven: no prior knowledge is needed for data pre-processing or pre-defined abnormal patterns; and (3) unified framework: *CAR* discovers top-k most relevant alerts and the corresponding alert patterns simultaneously, by modeling both temporal and content dependencies between alerts.

6 CONCLUSION

In this paper, we addressed an important and challenging problem of alert post-processing on massive heterogeneous categorical alert data. We proposed *CAR*, an unsupervised data-driven collaborative alert ranking algorithm to identify the true positive alerts and their associated abnormal patterns simultaneously. When tested on various enterprise cyber attack scenarios, *CAR* demonstrated the superiority in terms of various skill and robustness metrics, including 55 – 85% AUC improvement as well as interpretability of the detected alert patterns. An interesting direction for further exploration would be incorporating raw event data with the alert data in abnormal pattern generations.

REFERENCES

[1] Shyam Boriah, Varun Chandola, and Vipin Kumar. 2008. Similarity measures for categorical data: A comparative evaluation. In *Proceedings of the Eighth SIAM International Conference on Data Mining*. 243–254.

[2] Kevin Broughton. 2017. Automated incident response: respond to every alert. https://swimlane.com/blog/automated-incident-response-respond-every-alert/.

[3] Marco Caselli, Emmanuele Zambon, and Frank Kargl. 2015. Sequence-aware intrusion detection in industrial control systems. In *Proceedings of the Workshop on Cyber-Physical System Security*. 13–24.

[4] Varun Chandola, Arindam Banerjee, and Vipin Kumar. 2009. Anomaly detection: A survey. *Comput. Surveys* 41, 3 (2009), 15.

[5] Chao Chen, Daqing Zhang, Pablo Samuel Castro, Nan Li, Lin Sun, Shijian Li, and Zonghui Wang. 2013. iBOAT: Isolation-based online anomalous trajectory detection. *IEEE Transactions on Intelligent Transportation Systems* 14, 2 (2013), 806–818.

[6] Zhengzhang Chen. 2012. *Discovery of informative and predictive patterns in dynamic networks of complex systems.* Ph.D. Dissertation. North Carolina State University.

[7] Zhengzhang Chen, William Hendrix, and Nagiza F Samatova. 2012. Community-based anomaly detection in evolutionary networks. *Journal of Intelligent Information Systems* 39, 1 (2012), 59–85.

[8] Zhengzhang Chen, John Jenkins, Alok Choudhary, Jinfeng Rao, Vipin Kumar, Anatoli V. Melechko, Fredrick H. M. Semazzi, and Nagiza F. Samatova. 2013. Automatic detection and correction of multi-class classification errors using system whole-part relationships. In *SDM*. 494–502.

[9] Trevor Cohn, Phil Blunsom, and Sharon Goldwater. 2010. Inducing Tree-Substitution Grammars. *The Journal of Machine Learning Research* 11 (2010), 3053–3096.

[10] Oliver Dain and Robert K Cunningham. 2001. Fusing a heterogeneous alert stream into scenarios. In *Proceedings of the 2001 ACM Workshop on Data Mining for Security Applications*, Vol. 13.

[11] Kaustav Das and Jeff Schneider. 2007. Detecting Anomalous Records in Categorical Datasets. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 220–229.

[12] Boxiang Dong, Zhengzhang Chen, Hui (Wendy) Wang, Lu-An Tang, Kai Zhang, Ying Lin, Zhichun Li, and Haifeng Chen. 2017. Efficient discovery of abnormal event sequences in enterprise security systems. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management (CIKM'17)*. 707–715.

[13] FireEye. 2014. How many alerts is too many to handle. https://www2.fireeye.com/StopTheNoise-IDC-Numbers-Game-Special-Report.html.

[14] Jing Gao, Wei Fan, Deepak Turaga, Srinivasan Parthasarathy, and Jiawei Han. 2011. A spectral framework for detecting inconsistency across multi-source object relationships. In *Data Mining (ICDM), 2011 IEEE 11th International Conference on*. 1050–1055.

[15] Michael Gutmann and Aapo Hyvärinen. 2010. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *International Conference on Artificial Intelligence and Statistics*. 297–304.

[16] Alexander Hofmann and Bernhard Sick. 2011. Online intrusion alert aggregation with generative data stream modeling. *IEEE Transactions on Dependable and Secure Computing* 8, 2 (2011), 282–294.

[17] Glen Jeh and Jennifer Widom. 2002. SimRank: A measure of structural-context similarity. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 538–543.

[18] Klaus Julisch and Marc Dacier. 2002. Mining intrusion detection alarms for actionable knowledge. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 366–375.

[19] Charles L Lawson and Richard J Hanson. 1995. Solving least squares problems. *Society for Industrial and Applied Mathematics* 15 (1995).

[20] Aleksandar Lazarevic, Levent Ertöz, Vipin Kumar, Aysel Ozgur, and Jaideep Srivastava. 2003. A comparative study of anomaly detection schemes in network intrusion detection.. In *Proceedings of the 2003 SIAM International Conference on Data Mining*. 25–36.

[21] Wang Li, Li Zhi-tang, Li Dong, and Lei Jie. 2007. Attack scenario construction with a new sequential mining technique. In *Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing*, Vol. 1. 872–877.

[22] Xiaolei Li, Zhenhui Li, Jiawei Han, and Jae-Gil Lee. 2009. Temporal outlier detection in vehicle traffic data. In *2009 IEEE 25th International Conference on Data Engineering*. IEEE, 1319–1322.

[23] Alexander Y Liu and Dung N Lam. 2012. Using consensus clustering for multi-view anomaly detection. In *Security and Privacy Workshops (SPW), 2012 IEEE Symposium on*. IEEE, 117–124.

[24] Bo Long, Philip S Yu, and Zhongfei Zhang. 2008. A general model for multiple view unsupervised learning. In *Proceedings of the 2008 SIAM International Conference on Data Mining*. SIAM, 822–833.

[25] Chen Luo, Zhengzhang Chen, Lu-An Tang, Anshumali Shrivastava, Zhichun Li, Haifeng Chen, and Jieping Ye. 2018. TINET: Learning invariant networks via knowledge transfer. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD'18)*. 1890–1899.

[26] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*. 3111–3119.

[27] Xinzhou Qin and Wenke Lee. 2003. Statistical causality analysis of infosec alert data. In *International Workshop on Recent Advances in Intrusion Detection*. 73–93.

[28] Reza Sadoddin and Ali Ghorbani. 2006. Alert correlation survey: Framework and techniques. In *Proceedings of the 2006 International Conference on Privacy, Security and Trust: Bridge the Gap Between PST Technologies and Business Services*. 37.

[29] Makoto Shimamura and Kenji Kono. 2006. Using attack information to reduce false positives in network ids. In *11th IEEE Symposium on Computers and Communications*. 386–393.

[30] Georgios P Spathoulas and Sokratis K Katsikas. 2009. Using a fuzzy inference system to reduce false positives in intrusion detection. In *16th International Conference on Systems, Signals and Image Processing*. 1–4.

[31] Jiangwen Sun, Jinbo Bi, and Henry R Kranzler. 2014. Multi-view singular value decomposition for disease subtyping and genetic associations. *BMC Genetics* 15, 1 (2014), 73.

[32] Shiliang Sun. 2013. A survey of multi-view machine learning. *Neural Computing and Applications* 23, 7-8 (2013), 2031–2038.

[33] Jiliang Tang, Xia Hu, Huiji Gao, and Huan Liu. 2013. Unsupervised feature selection for multi-view data in social media. In *Proceedings of the 2013 SIAM International Conference on Data Mining*. 270–278.

[34] Esko Ukkonen. 1995. On-line construction of suffix trees. *Algorithmica* 14, 3 (1995), 249–260.

[35] Alfonso Valdes and Keith Skinner. 2001. Probabilistic alert correlation. In *International Workshop on Recent Advances in Intrusion Detection*. 54–68.

[36] Xin Zan, Feng Gao, Jiuqiang Han, and Yu Sun. 2009. A Hidden Markov Model based framework for tracking and predicting of attack intention. In *2009 International Conference on Multimedia Information Networking and Security*, Vol. 2. 498–501.

[37] Kai Zhang, Qiaojun Wang, Zhengzhang Chen, Ivan Marsic, Vipin Kumar, Guofei Jiang, and Jie Zhang. 2015. From categorical to numerical: Multiple transitive distance learning and embedding. In *Proceedings of the 2015 SIAM International Conference on Data Mining*. 46–54.