Automatically and Adaptively Identifying Severe Alerts for Online Service Systems

Nengwen Zhao[†‖], Panshi Jin[‡], Lixin Wang[‡], Xiaoqin Yang[‡], Rong Liu[§], Wenchi Zhang[¶] Kaixin Sui[¶], Dan Pei[*†‖]
[†]Tsinghua University [‡]China Construction Bank [§]Stevens Institute of Technology [¶]BizSeer
[‖]Beijing National Research Center for Information Science and Technology (BNRist)

***Abstract*—In large-scale online service system, to enhance the quality of services, engineers need to collect various monitoring data and write many rules to trigger alerts. However, the number of alerts is way more than what on-call engineers can properly investigate. Thus, in practice, alerts are classified into several priority levels using manual rules, and on-call engineers primarily focus on handling the alerts with the highest priority level (i.e., *severe alerts*). Unfortunately, due to the complex and dynamic nature of the online services, this rule-based approach results in missed severe alerts or wasted troubleshooting time on non-severe alerts. In this paper, we propose *AlertRank*, an automatic and adaptive framework for identifying severe alerts. Specifically, *AlertRank* extracts a set of powerful and interpretable features (textual and temporal alert features, univariate and multivariate anomaly features for monitoring metrics), adopts XGBoost ranking algorithm to identify the severe alerts out of all incoming alerts, and uses novel methods to obtain labels for both training and testing. Experiments on the datasets from a top global commercial bank demonstrate that *AlertRank* is effective and achieves the F1-score of 0.89 on average, outperforming all baselines. The feedback from practice shows *AlertRank* can significantly save the manual efforts for on-call engineers.**

I. Introduction

Large-scale and complex online service systems, such as Google, Amazon, Microsoft and large commercial banks, consist of thousands of distributed components and support a large number of concurrent users [1]. To maintain high-quality services and user experience, these companies use monitoring systems to collect various monitoring data from service components, for example, metrics/Key Performance Indicators (KPIs) [2], logs [1] and events [3]. Typically, engineers set up many rules to check the monitoring data and trigger alerts [4]–[6]. For example, if a selected metric exceeds a given threshold (e.g., CPU utilization exceeds 90%), an alert is generated to notify on-call engineers. Then an on-call engineer examines this alert. If this alert is severe, an incident ticket is created to initiate an immediate process of deep investigation and appropriate repair [7].

Unfortunately, in real world, these complex service systems generate a large number of alerts continuously, way more than what resource-constrained on-call engineers can properly investigate [5], [6], [8]. For example, Fig. 1 shows that thousands of alerts are generated per day in a large commercial bank C (China Construction Bank) [8]. Therefore, in practice, manual rules are used to classify alerts into different priority

[*] Dan Pei is the corresponding author.

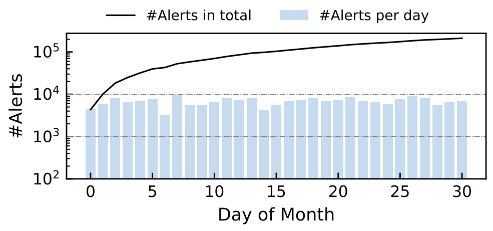


Fig. 1: The number of alerts in a large commercial bank in a given month (Y-axis is in logarithmic scale for clarity) [8].

levels, e.g., P1-critical, P2-error and P3-warning. These rules typically contain fixed threshold for KPIs (e.g., CPU utilization over 90% is P2-error, while exceeding 80% renders P3-warning), keywords matching for logs (e.g., different keywords like "warning", "error" and "fail" indicate different severities) and so on. In general, given the limited man power, on-call engineers mainly focus on the alerts with top priority (called **severe alerts** hereinafter), whose daily count can be still be in hundreds in bank C, before working on other alerts.

However, simple manual rules cannot sufficiently capture the patterns of complex and interacting factors that influence the priorities of the alerts. Furthermore, it is labor intensive for engineers to manually define and maintain rules, because 1) there are many types of alerts; 2) new types of alerts might be added due to system changes; 3) engineers might have different priority preferences [5]. As a result, in practice, above rule-based approach often results in missed severe alerts and lengthened time-to-repair, or results in wasted troubleshooting time on non-severe alerts. For example, in a dataset used in §IV from bank C, the precision and recall for identifying severe alerts is only 0.43 and 0.68, respectively.

To show the the damage of missed severe alerts, we present a real case in Fig. 2. There was a failure and related immediate P2 alert at 10:14 which indicated the monitored transaction response time increased to 500ms (exceeding a threshold specified in a P2 rule). However, the engineers are busy handling P1 alerts thus this alert was not immediately handled. The failure went unnoticed until customers called to complain at 10:45, wasting 31 minutes of repair time. Upon detailed investigation, multiple metrics (such as *success rate, transaction volume*) all experienced some anomalies during busy hours, based on which engineers thought there should has been a P1 alert, but there was no installed P1 rule that captured

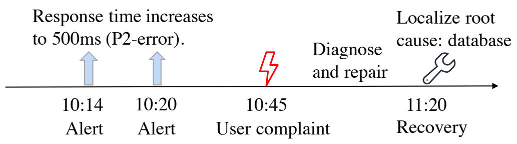


Fig. 2: A missing failure case due to the unsuitable rule-based strategy.

such a complex symptom. This case strongly demonstrates that multiple features are needed to determine the appropriate alert priorities.

Therefore, there is in an urgent need to design an effective machine learning-based algorithm that fully utilizes multiple features to identify severe alerts accurately, which can assist on-call engineers in differentiating high-priority severe alerts from non-severe alerts, in order to improve engineers' work efficiency as well as service quality. The challenges in designing such an algorithm are summarized as follows.

- Labeling overhead. With thousands of alerts arriving every day, it is tedious to manually label the severity of each alert.
- Large varieties of alerts. There are many types of alerts in practice, for example, application, network, memory and middleware. It would be labor intensive to manually define rules for determining the priority of each kind of alerts. An automatic approach can be extremely helpful.
- Complex and dynamic online service systems. Alert priority in such a complex system is influenced by various factors. Besides, large-scale online services are often under constant change, e.g., configuration change, software upgrade, which may add new alerts. Therefore, the approach should be adaptive to stay in tune with the dynamic environment.
- Imbalanced data. In general, only a small portion of alerts are considered severe. For example, in our experimental datasets, the ratio between non-severe and severe alerts is around 50:1 (§IV-A). Learning (e.g., binary classification) from imbalanced data can be another challenge.

To tackle the above challenges, we propose an automatic and adaptive framework named *AlertRank* for identifying severe alerts. *AlertRank* includes two components, offline training and online ranking. In offline training, to handle complex online service, *AlertRank* extracts a set of interpretable features from historical alerts and monitoring metrics to characterize the severities of alerts. The resolution records in historical alerts are grouped into a very small number of clusters, each of which is labeled with a severity score by engineers. Then each historical alert is *automatically* assigned a severity score (§III-A2), avoiding manual labeling.

Instead of simply classifying whether an alert is severe or not, *AlertRank* formulates the problem of identifying severe alerts as a ranking model, which can handle class imbalance and is user-friendly for engineers. Then the XGBoost ranking is utilized to train a ranking model [9]. Besides, an incremental training pipeline is leveraged to make our model adapt to dynamic environment. During online ranking, when alerts arrive, *AlertRank* extracts the features from current data and apply the trained ranking model to prioritize these alerts based on the output severity scores, so that engineers can investigate the arrived alerts based on the guidelines given by *AlertRank*.

The major contributions of this paper are the following:

- To the best of our knowledge, this paper is the first one that proposes an automatic and adaptive approach for identifying severe alerts for online service systems.
- We design a comprehensive set of features from multiple data sources to handle complex online services. These features include textural (topic [10] and entropy [11]) and temporal features (frequency, seasonality [12], count, inter-arrival time [13]) from alerts, and univariate and multivariate anomaly features from metrics [14], which are interpretable and can further help engineers diagnose incidents.
- Different from traditional classification models, we formulate the problem of identifying severe alerts as a ranking model, which can handle class imbalance and instruct engineers to repair which alert first [15].
- We novelly proposes to use historical tickets as the testing labels, and propose to use clustering-based approach (with only cluster-level labels) to automatically assign severity score labels for all alerts as the training labels for ranking.
- Experiments on real-world datasets show *AlertRank* is effective with a F1-score of 0.89 on average. Furthermore, our real deployment demonstrates *AlertRank* can significantly save the manual efforts for on-call engineers.

II. Background

A. Alert Management System

Fig. 3 provides a high-level summary of an IT operations ecosystem consisting of service architecture, monitoring system, and alert management system [3]. The monitoring system collects various data (e.g., KPIs, logs and events) continuously from service components. To ensure service availability and stability, service engineers manually define many rules to check monitoring data and trigger alerts [5]. With alerts received, the alert management system processes them through some common techniques, such as alert aggregation and correlation, to filter duplicated and invalid alerts. For the remaining alerts, as introduced in §I, on-call engineers usually first examine alerts with high priorities (P1). If an alert is severe and hard to repair in time, an incident ticket is created [7]. However, as mentioned in §I, the rule-based severity classification may fail in practice. In this paper, we tackle the critical problem of identifying severe alerts accurately and adaptively, so as to assist engineers in fixing true severe alerts responsively and preventing potential failures.

B. Data Description

1) Alert: Alert data, as the main data objects in our study, have multi-dimensional attributes. Table I presents an example alert with several major attributes. "Content" specifies the detailed information of the alert, and "Resolution Record" records how the alert was resolved in detail, which usually written by on-call engineers or generated by system automatically. "Resolution Record" can be used to label a severity score for each alert, which will be introduced in §III-A2. We

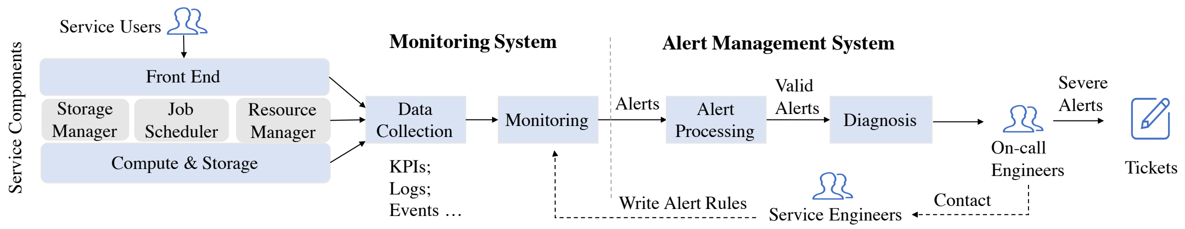


Fig. 3: Summary of the service architecture, monitoring system, alert management system and different people involved [3].

will extract alert textual and temporal features from "Content" and "Time" attributes respectively (§III-B).

TABLE I: An example alert.

Time	Severity	Type
2019-02-20 10:04:32	P2-error	Memory
AppName	**Server**	**Close Time**
E-BANK	IP(*.*.*.*)	2019-02-20 10:19:45
Content		
Current memory utilization is 79% (Threshold is 60%).		
Resolution Record		
Contact the service engineers responsible for E-BANK and get a reply that there is no effect on business, then close the alert.		

2) Key Performance Indicator: KPIs (Metrics) are another crucial data type in service management, which are collected continuously at a fixed time interval, e.g., every minute, to measure the health status of servers (e.g., CPU utilization) or business (e.g., response time) [2], [16]. KPI anomalies (e.g., sudden spike or dip) may indicate some problems about business and servers. In addition to alert data, we also extract KPI anomaly features from important business KPIs and server KPIs to characterize the overall status of the system (§III-C).

3) Ticket: In general, tickets are created either from alerts or from failures. When investigating an alert, if engineers find it is hard to fix the alert in time, or the alert has a significant business impact, they create a ticket from this alert so as to follow up with a repair process [7], [17], [18]. On the other hand, failures, e.g., the case in Fig. 2, are critical in service management, because they directly affect service stability. A failure is usually reported by users or discovered by service engineers, instead of being reported by any alerts in advance. Engineers will create a ticket and repair the failure quickly. Often, when diagnosing a failure and localizing its root cause, engineers may find some early alerts which have been ignored by mistake (e.g., alert in Fig. 2). Thus, we can determine the alert is severe or not by whether it is associated with a ticket.

III. Design

A. Overview

1) AlertRank Overview: Fig. 4 illustrates the architecture of *AlertRank*. In the offline learning module, inspired by the idea of multi-feature fusion, we carefully extract a set of interpretable features from historical alerts (textural and temporal features) and KPIs (univariate and multivariate anomaly features), based on domain knowledge in our context. Then a ranking model is constructed with the popular XGBoost ranking algorithm [9]. The severity scores used for training

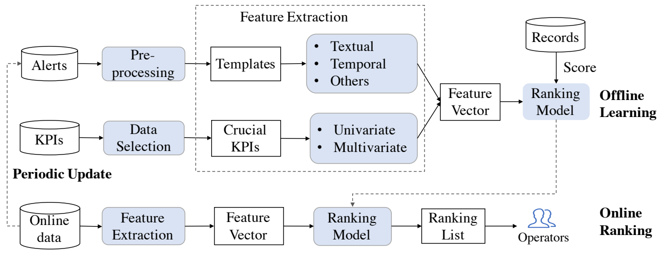


Fig. 4: *AlertRank* overview.

are obtained by resolution records in Table I, which will be introduced in §III-A2 in detail. In the online ranking, the features extracted from incoming alerts and corresponding KPIs are directly fed into the trained ranking model. The model ranks the incoming alerts based on the output severity scores, so that on-call engineers can first examine alerts with higher severity scores. The alerts whose severity scores exceed a specific threshold is considered severe and the threshold is chosen based on the performance on training set. In addition, to make our model adaptive to the dynamic IT environment, we build an incremental training pipeline where our model is trained with latest data periodically so that changes can be captured properly in time.

2) Automatic Labeling: As discussed in §I, labeling overhead is a big challenge for our problem. In *AlertRank*, we novelly utilize the historical tickets and resolution records to obtain the labels without manual efforts.

Binary label. The problem which this paper aims to solve is identifying severe alerts. Thus we need give each alert a binary label (severe/non-severe) for evaluation (§IV-B). As stated in §II-B3, each alert can be labeled as severe or non-severe according to whether it is associated with a ticket.

Continuous label. To train the ranking model, we also label each alert with a specific severity score (between 0 and 1) based on its resolution record, which is more comprehensive than simple binary label. It is because that the real severity of each alert varies in all severe/non-severe alerts. As stated in §II-B1, resolution records are written by on-call engineers or generated by system automatically. In general, there are several different types of resolution records which indicate different severities. We adopt TF-IDF vectorization [11] and k-means [19] to cluster resolution records, and the value of k is determined by silhouette coefficient (k=7) [20]. The clustering centroids are presented in Fig. 5 and experienced engineers give a severity score for each cluster. In this way, each alert can be automatically assigned a severity score based on its

1. None. (0; 65.1%)
2. This alert is in white list. (0.1; 4.2%)
3. This alert has been recovered automatically. (0.2; 7.8%)
4. Contact the service engineers and there is no effect on business. (0.4; 10.6%)
5. Known reasons. This alert has been resolved. (0.6; 6.4%)
6. Contact the service engineers and there is an effect on business. Already resolved. (0.8; 3.8%)
7. Create a ticket. (1; 2.1%)

Fig. 5: The clustering centroids of dataset in bank C and the corresponding (severity score; percentage).

resolution record, so as to save considerable manual efforts. Besides, during our study of the 18-month-long dataset, we observe that the patterns of resolution records generally remain unchanged unless new on-call engineers are involved.

B. Alert Features Extraction

1) Alert Preprocessing: Before extracting alert features, we first preprocess alert data as follows.

Tokenization. The textual descriptions of alerts ("Content" in Table I) usually combine words and symbols (e.g., punctuation). Tokenization filters out these symbols and divides the remaining text into tokens. We further remove stop words, those highly frequent words like "to", "are" and "is", since they are useless in identifying severe alerts.

Alert parsing. The description of an alert is semi-structured text generated by the monitoring system with two types of information: a constant string that describes an abstract alert event, and parameters that record some system variables (e.g., IP address, KPI value, file path, instance name, etc.). A common practice is parsing alerts which extracts the constant string to form an alert template. For example, in Fig. 6, alerts A1 and A4 can fit into the template T1 and T2, respectively. The remaining parts are variables (the underscored text). Note that it does not matter to ignore some KPI values (e.g., 67% and 73%) since they are reflected in the rule-based severity ("Severity" in Table I).

Parsing methods have been well studied in log data but have not been applied in alerts. We adopt FT-tree [21], one of the state-of-the-art log parsing methods, to match each alert to a specific alert template. Based on the key observation that a "correct" alert template is usually a combination of words that occur frequently in alerts, FT-tree dynamically maintains a tree structure of such frequent words that implicitly defines the set of alert templates. In addition to high accuracy, FT-tree is naturally incrementally retrainable, because this implicitly-defined set of alert templates dynamically and incrementally evolves with the arrival of new types of alerts (e.g., due to the aforementioned software upgrades) [21].

2) Textual Feature: After alert preprocessing, the semi-structured alerts are transformed into normalized templates. Then we extract some textual features from the alert templates.

Topic. Intuitively, each alert template can be regarded as a document describing one or more topics of IT operations. We can apply a topic model [22] to extract the hidden semantic features. Many topic models have been proposed in the literature, for example, Latent Dirichlet Allocation (LDA) [23]. A conventional topic model, such as LDA, typically works

Alert Contents:
A1: Memory utilization current value is 67%. It exceeds the threshold.
A2: TCP CRITICAL - 0.7 second response time on port 3306.
A3: The number of processes is abnormal (instance: TimeoutCtrl), current value is 0.
A4: TCP CRITICAL - 0.8 second response time on port 3302.
A5: Memory utilization current value is 73%. It exceeds the threshold.

Alert Templates:
T1: Memory utilization current value is *. It exceeds the threshold.
T2: TCP CRITICAL - * second response time on port *
T3: The number of processes is abnormal (instance: *), current value is *.

Fig. 6: Explanation of alert template extraction.

well with documents consisting of rich text, while our alert text is usually very short. Thus, we adopt Biterm Topic Model (BTM) [10], an algorithm designed specifically for short texts, which has shown better performance than LDA in short texts. BTM explicitly models the word co-occurrence patterns (i.e., biterms), rather than documents, to enhance the topic learning [10]. Given a predefined number of topics, BTM discovers hidden topics and keywords corresponding to each topic. The number of topics in our problem is selected based on coherence score (n_topics=14), which is a metric to evaluate the quality of extracted topics [24]. Fig. 7 presents some examples of extracted topics and corresponding keywords from our experimental dataset. For example, we can infer that $T\#1$ and $T\#2$ are related to oracle database and syslog, respectively. BTM can output the probability that an alert belongs to each topic. Fig. 8(a) shows the relationship between alert topics and severity scores, and we can observe that the alerts belonging to different topics indicate different severities. Considering that an alert may have mixture topics, we utilize the output probabilities as topic features.

T#1: oracle, connection, database, space, pool, process, lock
T#2: syslog, alert, error, stack, records, hardware, warning
T#3: monitor, environment, server, temporal, battery, power, voltage
…
T#13: unaccessible, export, response, packet, password, order, accounting
T#14: switch, virtual, communication, connection, health, network, report

Fig. 7: Extracted topics by BTM and some corresponding representative keywords.

Entropy. Considering that alert is a bag of words and different words have different importance (entropy) in identifying severe alerts. For example, the word "timeout" is more informative than "port". Therefore, in addition to topic features, we also consider the entropy of each alert template. IDF (Inverse Document Frequency) is widely utilized in text mining to measure the importance of words, which downgrades frequent words while increasing the weights of rare words [11]. For each word, the IDF value is calculated as $IDF(w) = \log \frac{N}{N_w+1}$, where N refers to the total number of alerts and N_w is the number of alerts containing the word w. Intuitively, if a word frequently appears in historical alerts, its discriminative power is much lower than the one that only appears in a small number of alerts [1]. Based on the IDF of each word calculated from the training data, we can compute the entropy of each alert as $\frac{\sum_w IDF(w)}{\#w}$, where $\#w$ is the number of words in this alert. Fig. 8(b) shows the alerts with

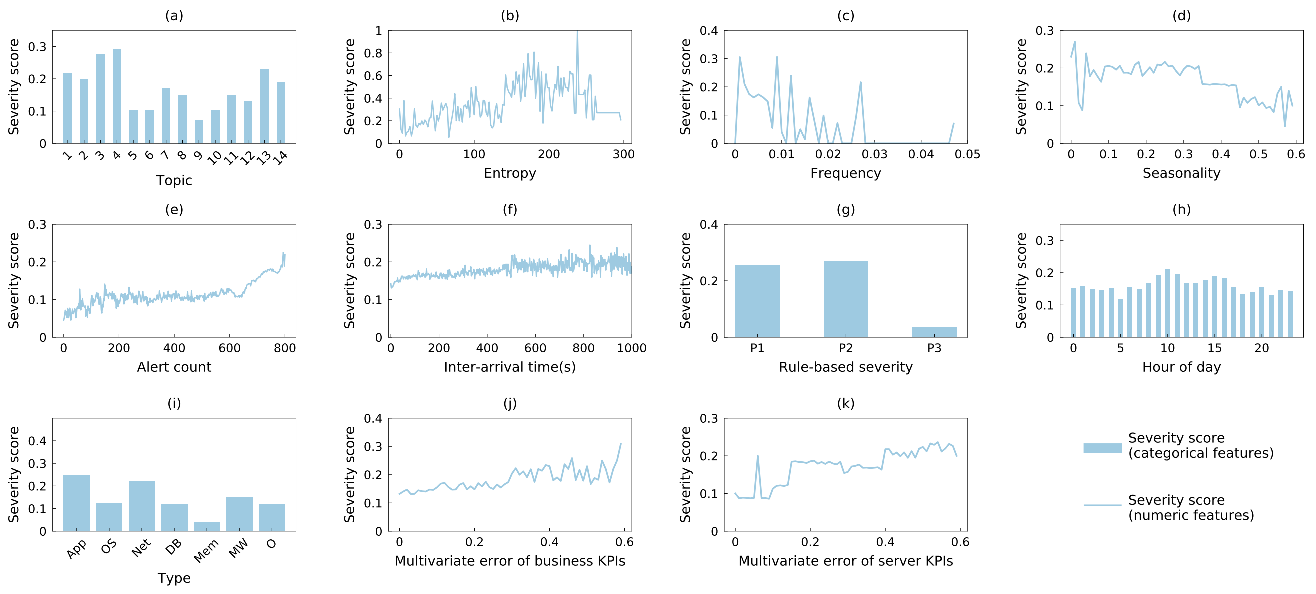


Fig. 8: The qualitative relationship between severity scores and some representative features. The severity scores are automatically obtained based on resolution records (§III-A2).

high entropies are more likely to be severe.

3) Temporal Feature: In addition to textual features, temporal features derived from alert time should also be considered.

Frequency. Some alerts, such as CPU overload, occur frequently, while others, such as device outages, occur rarely. Fig. 8(c) demonstrates the relationship between frequencies and severity scores. In general, alerts with low frequencies are more likely to be severe. Thus, the frequency of each alert can be indicative to determining whether an alert is severe.

Seasonality. Based on our observation, some alerts occur quasi-periodically. For example, running a batch task every night may cause CPU and disk alerts. Intuitively, seasonal alerts are less informative than irregular ones. For one alert a, we obtain a time series $C(a) = \{c^1(a), c^2(a), \cdots, c^h(a)\}$, where $c^k(a)$ is the number of occurrences of alert a in the k-th time bin and h is the number of time bins. Here the time bin is set to 15 minutes. Clearly, if a is a seasonal alert, $C(a)$ is a periodic time series.

In our approach, we adopt Autocorrelation Function (ACF), one of the most popular time series periodicity detection techniques, to characterize the periodicity of the alert [25], [26]. In detail, given a time series x with length N and different lags l, we have

$$ACF(l) = \frac{\sum_{i=0}^{N-1} x(i)x(i+l)}{N}, l = 1, 2, \cdots, \tfrac{N-1}{2} \quad (1)$$

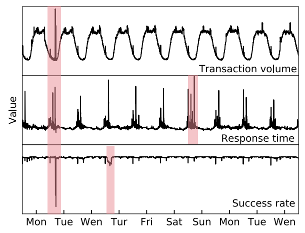
It is clear that if the time series is periodic with length T, the autocorrelation becomes high at certain lags, i.e., $T, 2T, 3T, \cdots$. Therefore, a larger $ACF(l)$ implies a stronger seasonality. We use the maximum value of $ACF(l)$ as the final seasonality feature. We can observe from Fig. 8(d) that larger seasonality value tends to have low severity score.

Alert count. We define alert count as the number of alerts that occur during a time window (e.g., 30 minutes before the current alert) [13]. Intuitively, engineers need to pay more attention to alerts that burst within a short time window (Fig. 8(e)). Besides, we also consider the numbers of alerts with different severity levels during this time window.

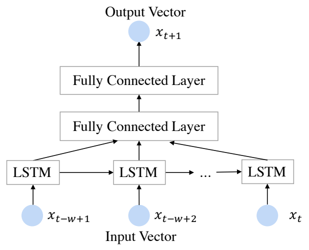
Inter-arrival time. The inter-arrival time is defined as the time interval between an alert and its proceeding one [13]. Clearly, if an alert suddenly breaks out after a system has been running without alerts for a long time (long inter-arrival time), it may be severe and need more attention (Fig. 8(f)).

4) Other Features: In addition to these carefully designed features described above, we also adopt some simple features directly from the attributes of alert data (Table I).

- Rule-based severity. Fig. 8(g) shows the relationship between the real severity scores and rule-based severity levels. It is intuitive that both P1 and P2 alerts tend to have higher severity scores, thus it is unreasonable to focus only on P1 alerts while ignoring P2. However, we can still utilize the original severity as a feature.
- Alert time. Clearly, the alert time has an influence on the importance of the alert. For example, as depicted in Fig. 8(h), alerts that occur during busy hours (9:00-11:00 and 14:00-16:00) are more severe compared with those that occur at night. Therefore, we adopt three features to characterize the occurrence time of an alert, i.e., during busy hours or not, day or night, weekday or weekend.
- Type. We also notice that alert types have impact on their severity levels. Fig. 8(i) shows the average severity scores under several different types of alerts (application, operating system, network, memory, middleware and others). It is evident that application alerts tend to have higher severity scores, perhaps because these alerts are more closely related to service reliability and user experience.



(a) An example of business KPIs.



(b) LSTM model.

Fig. 9: Illustration of KPI features extraction.

C. KPI Features Extraction

In addition, since KPIs can characterize the health status of servers and applications [27], we incorporate KPIs to capture the alert severity better. Note that although business KPIs and server KPIs can directly trigger alerts, these alerts alone cannot describe the health status of systems accurately, because they are generated based on simple threshold rules. Thus, in this study, we try to design a more accurate and generic method to capture the KPI anomalies. In particular, we believe attention should be given to those alerts that co-occur with anomalies of important business KPIs or server KPIs.

However, usually a number of KPIs are measured to monitor a service, and it is time consuming for engineers to check all KPIs manually. In our approach, we choose some representative business KPIs (response time, success rate, transaction volume, processing time) and server KPIs (CPU utilization, I/O wait, memory utilization, load, network packets, the number of process, disk I/O) which are closely associated with service availability and user experience, and adopt a multi-KPIs anomaly detection technique to accurately measure the overall status of business and servers. Fig. 9(a) presents an example of three important business KPIs of a service, and the pink bands denote the anomalies.

We utilize the state-of-the-art multi-variate time series anomaly detection algorithm based on LSTM [14], [28]. Compared to traditional recurrent neural networks (RNNs), LSTM has an improved capability to maintain the memory of long-term dependencies, because it uses a weighted self-loop to accumulate or forget past information conditioned on its context [14]. Thus, it can learn the relationship between past and current data values, and has demonstrated remarkable performance in various sequential data [29]. Fig. 9(b) presents the LSTM structure in our approach. The input vector is the time window $\{x_{t-w+1}, x_{t-w+2}, \cdots, x_t\}$, where x_i is an m-dimension vector denoting the value of each important KPI at time i (e.g., m=3 in Fig. 9(a)). The goal is to predict the vector x_{t+1} and the prediction error can be used to characterize the degree of anomaly. In our model, we use the overall prediction error and also the error on each dimension (univariate and multivariate) as features. In Fig. 8 (j) and (k), we plot the relationship between severity scores and the normalized overall prediction errors of business KPIs and server KPIs, respectively. We can observe that higher prediction errors indicate higher severity scores.

TABLE II: Features used in *AlertRank*.

Feature Type	Feature Name	#Feature
Alert Textural	BTM Topics (14), Entropy (1)	15
Alert Temporal	Frequency (1), Seasonality (1), Alert count (4), Inter-arrival time (1)	7
Alert Attributes	Original severity (1), Alert time (3), Type (1)	5
KPI Anomaly	Univariate anomaly (11), Multivariate anomaly (2)	13

Feature Engineering Summary. In conclusion, Table II summarizes a total of 40 features adopted in *AlertRank*. We design these features based on careful data analysis and discussion with experienced engineers, thus these features are associated with rich domain knowledge. If *AlertRank* will be applied in other scenarios, some new features may need to be introduced into our model, but the core idea and pipeline of *AlertRank* is generic. Based on our feature engineering study, we find that the relationships between alert severity and various features are remarkably complex (Fig. 8). This also explains why a simple rule-based strategy cannot identify severe alerts accurately. For example, the alert in Fig. 2 is classified into P2, but KPI anomaly features and alert time will increase the severity of this alert. The effectiveness of each kind of features will be demonstrated in §IV-C2.

D. Ranking Model

AlertRank proposes a machine learning framework to incorporate various features based on data fusion and learn a ranking model to identify severe alerts. We novelly formulate this problem as a ranking model, instead of binary classification, due to the following reasons. First, dichotomous results given by classification (severe/non-severe) are prone to false positives or false negatives. In particular, false negatives make some potential failures undetected, leading to downgraded service availability. However, ranking model can prioritize alerts based on severity scores and guide engineers to repair which alert first, which is more user-friendly. Second, ranking model can deal with class imbalance effectively [15]. The effectiveness of ranking model will be presented in §IV-C3.

To train a ranking model, as introduced in §III-A2, each alert is labeled a severity score automatically based on its resolution record. There are three common ranking approaches, namely, pointwise, pairwise and listwise [30]. In our problem, the training data naturally comes as pointwise (each alert has a severity score). Besides, the other two ranking approaches (e.g., LambdaMART [31]) only provide the relative order of alerts, and cannot output a specific severity score for each alert. Therefore, we adopt the XGBoost pointwise ranking algorithm based on regression trees [9]. XGBoost is a gradient boosting tree based model that is widely used by data scientists and achieves state-of-the-art performance on many problems [9].

In real practice, the alert management system receives many streaming alerts continuously, and on-call engineers tend to investigate alerts in batch at a regular interval (e.g., every 15 minutes). Therefore, during online testing, the trained ranking model is applied to prioritize these incoming alerts based on the output severity scores every 15-minute interval. An alert is considered as severe if its predicted severity score exceeds a threshold, and the threshold is properly selected based on its best performance on training set. In general, engineers first examine the alerts with higher severity scores. Furthermore, as we mentioned in §I, in order to adapt to dynamic online services, the ranking model is incrementally trained with the latest training data periodically.

IV. Evaluation

In this section, we evaluate our approach using real-world data and aim to answer the following research questions:

- RQ1: How effective is *AlertRank* in identifying severe alerts?
- RQ2: How much can the alert features and KPI features contribute to the overall performance?
- RQ3: Is the ranking model adopted in *AlertRank* effective?
- RQ4: Is the incremental training pipeline useful?

A. Datasets

To evaluate the performance of *AlertRank*, we collect three real-world datasets named A, B and C from a top global commercial bank through its alert management system. Table III summarizes these datasets. Each dataset has a time span of six months. These alerts have been cleansed by the alert processing module in Fig. 3 to filter out duplicate and invalid one. As we mentioned in §III-A2, each alert is labeled a binary class (severe/non-severe) by tickets for evaluation and a severity score by alert resolution records for training ranking model. We observe that each dataset contains an imbalanced mixture of severe and non-severe alerts, with a ratio about 1 : 50. In our experiments (RQ1-RQ3), for each dataset, we use alerts occurred in the first five months as training set, and the last one month without new alerts as testing set.

TABLE III: Details about the experimental datasets.

Datasets	Time span	#Alerts	#Severe alerts
A	2018/01/01∼2018/06/30	374940	7012
B	2018/07/01∼2018/12/30	429768	8482
C	2019/01/01∼2019/06/30	390437	7445

B. Metric

As stated in §III-D, the trained ranking model is applied to rank incoming alerts based on the output severity scores at a regular interval (e.g., 15 minutes). For each interval, an alert is considered as severe if its severity score exceeds a threshold. We properly choose the threshold that can maximize the performance on training set. Therefore, precision/recall/F1-score are calculated for evaluation. Precision measures the percentage of identified severe alerts that are indeed severe. Recall measures the percentage of severe alerts that are correctly identified. F1-score is the harmonic mean of precision and recall. We take the average metrics of all intervals as the final metrics. Besides, in order to evaluate the ranking capability of *AlertRank* (RQ3), we also compute another metric, precision@k, i.e., the precision rate of top-k alerts [15].

C. Evaluation Results

1) RQ1: **How effective is *AlertRank* in identifying severe alerts?**

In order to demonstrate the effectiveness of *AlertRank*, we compare it with two baseline methods.

- Rule-based. As introduced in §I, in practice, alerts are separated into several severity levels (e.g., P1-critical, P2-error and P3-warning) based on rules. Usually, engineers mainly focus on P1 alerts while ignoring P2 and P3 alerts.
- Bug-KNN [32]. Identifying severe bug reports is a critical issue in software engineering. Bug-KNN utilizes K-Nearest Neighbor (KNN) to search for historical bug reports that are most similar to a new bug through topic and text similarity.

TABLE IV: Performance comparison between *AlertRank* and two baseline methods.

Datasets	A			B			C		
Methods	P	R	F1	P	R	F1	P	R	F1
AlertRank	**0.85**	**0.93**	**0.89**	**0.82**	**0.90**	**0.86**	**0.93**	**0.92**	**0.93**
Rule-based	0.43	0.68	0.53	0.47	0.70	0.56	0.41	0.74	0.53
Bug-KNN	0.72	0.76	0.74	0.79	0.62	0.70	0.80	0.53	0.64

Table IV shows the precision (P), recall (R) and F1-score (F1) comparison between *AlertRank* and baseline methods. Clearly, *AlertRank* exhibits outstanding performance on all datasets and achieves the F1-score of 0.89 on average, higher than other baseline methods. The results indicate that traditional rule-based strategy is not effective, because whether an alert is severe is influenced by various factors and simple rules cannot capture all these factors. Besides, due to system complexity, it is difficult to set suitable rules, and each engineer may have his or her own preference and knowledge to set rules [5]. In terms of Bug-KNN, it determines severe alerts based on only the textual description of alerts. Clearly, *AlertRank*, which incorporates both alert features (textual and temporal) and KPI features achieves better performance. Furthermore, KNN is a lazy learner because it does not learn a discriminative function from the training data but computes the similarity between a new alert and all historical alerts. It suffers from extremely high computational complexity.

In summary, the results show that *AlertRank* is effective in identifying severe alerts for online service systems.

2) RQ2: **How much can the alert features and KPI features contribute to the overall performance?**

AlertRank incorporates two types of features: alert features and KPI features. In this RQ, we evaluate the effectiveness of each type and the results are presented in Table V. Our model can achieve an average F1-score of 0.89 with high precision and recall, when both types of features are used. With the alert features alone, the average F1-score drops from 0.89 to 0.76. However, if only the KPI features are used, the average F1-score drops dramatically from 0.89 to 0.36. This indicates that our model benefits from the ensemble features extracted from multiple data sources and achieves the best overall results. Besides, the results also indicate that the alert features are more powerful than KPI features.

TABLE V: Effectiveness of the ensemble features.

Datasets	A			B			C		
Methods	P	R	F1	P	R	F1	P	R	F1
AlertRank	**0.85**	**0.93**	**0.89**	**0.82**	**0.90**	**0.86**	**0.93**	**0.92**	**0.93**
Alert Only	0.82	0.79	0.80	0.75	0.80	0.77	0.67	0.77	0.72
KPI Only	0.42	0.40	0.41	0.32	0.39	0.35	0.36	0.31	0.33

3) *RQ3*: **Is the ranking model adopted in *AlertRank* effective?**

As stated in §III-D, we formulate the problem of identifying severe alerts as a ranking model. In order to illustrate the effectiveness of ranking model, motivated by [15], we compare our model with three popular classification models, i.e., SVM, Random Forest (RF) [33] and XGBoost [9]. Weighted classification is a common approach to deal with class imbalance and we adopt it to assign a larger weight to severe alerts. The classification threshold is also selected based on its best performance on training set.

Table VI shows the F1-score comparison and it is evident that XGBoost ranking outperforms the classification models. It is because that weighted classification which aims to learn a discriminative function from the binary label is still a little sensitive to class imbalance. Besides, the true severity of each alert varies in all severe/non-severe alerts, while the binary label is insufficient for model learning. However, the ranking model which is trained with comprehensive severity scores can mitigate the problem of imbalanced severe alerts.

In order to show the ranking ability of *AlertRank*, we rank the probabilities returned by each classification algorithm [15]. We choose the time intervals with more than two severe alerts in testing set and compute the precision@k. As shown in Fig. 10(a)-10(c), *AlertRank* can effectively rank the severe alerts and consistently achieve high precision. More importantly, we compare the precision under no false negatives as shown in Fig 10(d). It is clear that with 100% recall, *AlertRank* achieves a much higher precision than other methods. In other words, with *AlertRank*, the number of alerts which need examined by engineers will be reduced by about 20%, while ensuring no single severe alert is missed.

In summary, the ranking model adopted in *AlertRank* is indeed effective compared with classification models.

TABLE VI: F1-score comparison between XGBoost ranking model and other classification methods.

Datasets	SVM_c	RF_c	XGBoost_c	***AlertRank***
A	0.75	0.79	0.80	**0.89**
B	0.70	0.77	0.78	**0.86**
C	0.80	0.81	0.85	**0.93**

4) *RQ4*: **Is the incremental training pipeline useful?**

An online service system in real world is under constant change due to new applications deployment, software upgrade, or configuration change, etc. New alert types and rules are also added accordingly. To address this issue, as introduced in §III-A, *AlertRank* incorporates an incremental training strategy. This RQ aims to evaluate the effectiveness of the incremental training.

We choose dataset C in this RQ, because there is a major software change on April 19, resulting in many new alerts emerging after that day. The data in the first three months is used as the training set (2019/01/01∼2019/03/31) and the remaining as our testing set (2019/04/01∼2019/06/30). We conduct three experiments. In one experiment, the trained model is applied to the testing data directly without incremental update. In the other two experiments, we update the model incrementally every day and every week, and the threshold is updated accordingly. Besides, XGBoost based on gradient boosting tree can friendly support incremental training.

Table VII presents the precision/recall/F1-score comparison before and after the software change. Clearly, before April 19, the alert patterns on the testing set are similar to the training set. Thus, incremental update has no impact on model performance. However, after the software change, we can observe that the model that was trained with past data does not always perform well in the future. However, the incremental training improves the F1-score significantly, from 0.68 to 0.88. Moreover, we find that the daily model update delivers slightly better performance than the weekly update.

In summary, incremental training is indeed essential to keep our models in tune with highly dynamic online systems.

TABLE VII: Effectiveness of incremental training.

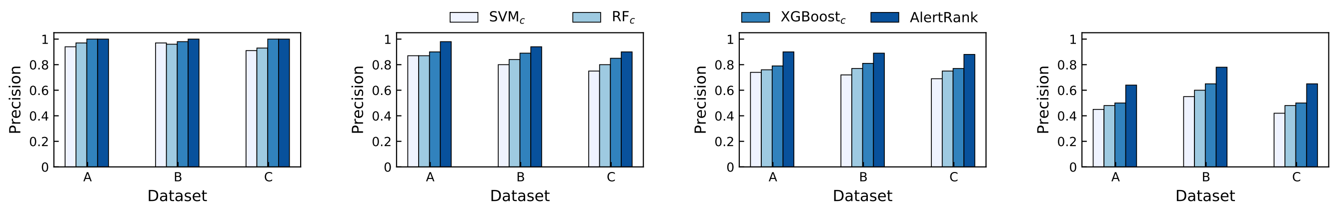
Methods	W/o update			Weekly update			**Daily update**		
Time	P	R	F1	P	R	F1	P	R	F1
∼ Apr.19	0.82	0.87	0.84	0.83	0.87	0.85	**0.85**	**0.89**	**0.87**
Apr.19 ∼	0.76	0.62	0.68	0.80	0.82	0.81	**0.87**	**0.89**	**0.88**

V. Operational Experience

A. *Success Story*

We have successfully applied *AlertRank* on the alert management platform in a top global commercial bank. The deployment shows that *AlertRank* can identify severe alerts more accurately than rule-based method. For example, about the alert in Fig. 2 which is classified into P2 by mistake, *AlertRank* can accurately identify this severe alert because business KPIs anomaly features and alert time (during busy hour) increase the severity of this alert.

To demonstrate *AlertRank*'s operational excellence quantitatively, we count the number of alerts that need to be manually examined under the rule-based strategy and *AlertRank* perspectively. For rule-based method, engineers usually only examine P1 alerts and ignore others. For *AlertRank*, as stated in §III-A, engineers only check the alert whose severity scores are higher than the threshold which is selected based on best performance on training set (0.86, 0.82, 0.87 for dataset A, B and C, respectively). Table VIII shows the effort reduction achieved by *AlertRank* on testing set (one month). Clearly, using traditional rule-based strategy, engineers need to investigate more alerts, and they waste much time in investigating non-severe alerts (low precision) but still miss many severe alerts (not high recall). In comparison, *AlertRank* can significantly reduce the number of alerts which engineers



(a) Precision@1. (b) Precision@2. (c) Precision@3. (d) Precision (when recall=1).

Fig. 10: Demonstration of the ranking ability of *AlertRank*.

need to investigate, while ensuring high precision and recall. Therefore, *AlertRank* can save the manual effort for on-call engineers significantly while retaining alert quality.

TABLE VIII: Comparison of the number of alerts that need to be investigated (#alerts) and precision/recall (P/R).

Datasets		A	B	C
Rule-based	#Alerts	1996	2536	2094
	P/R	0.43/0.68	0.47/0.70	0.41/0.74
AlertRank	#Alerts	1380	1869	1148
	P/R	0.85/0.93	0.82/0.90	0.93/0.92

B. Efficiency

In order to embed our framework in the alert management platform to help engineers detect severe alerts in real time, we carefully evaluate the response time of *AlertRank*. Based on our experiments, *AlertRank* can rank 100 alerts in about 2.4 second, which is very acceptable to engineers. It is because the same alert templates can share the textual features and some temporal features extracted in training set, which save much time in feature extraction when online ranking. Besides, the ranking model can be trained offline with historical data and then be applied to online data directly. It takes about twenty minutes to train a model with three-month data on Intel Xeon CPU E5-2620 machine with 10 cores.

VI. Related Work and Limitation

A. Related Work

Despite a great deal of efforts have been devoted into alert management, including aggregation [34], correlation [35], [36] and clustering [37], effective approaches to identifying severe alerts have remained elusive. [5] proposed a simple alert ranking strategy based on the linear relationships between alert thresholds. However, the assumption of a linear relationship between two KPIs may not always hold in reality. [6] proposed to learn a rule-based strategy from historical data to distinguish real alerts from non-actionable alerts. Both of the two methods only consider the KPIs alerts and revise the threshold strategy. However, there are various of alerts in practice, such as network, database and logs. Besides, the severity of an alert is influenced by various factors, but these rule-based methods cannot capture the complex relationships of these factors.

Identifying severe alerts is also a relevant research topic in the fields of intrusion detection system (IDS) [8], [38], [39] and software engineering [32], [40]. Specifically, IDS prioritizes alerts based on the domain knowledge about security (e.g., the source IP and destination IP) and the attack path, which is different from our scenario. Identifying severe bug reports is a critical issue in software engineering. Most methods in this area aim to extract some textual features (e.g., frequency and emotion) from bug reports [32], [40], then apply machine learning techniques to determine the bug severity. In our scenario, alert severity is much more complicated than bugs, because we need to consider various factors due to the complex and highly dynamic online service environments. Thus these methods do not perform well, which has been demonstrated in §IV-C1.

B. Limitation

One limitation of this paper is we do not provide too much details about parameter selection (e.g., the number of topics, the number of resolution record clusters, etc), because of the limit of space. Research on the influence of different parameters on algorithm performance can be our future work. Besides, more reasonable and generic labeling method, incorporating feature selection technique and selecting the decision threshold more adaptively can also be our future work.

VII. Conclusion

Large-scale online service systems generate an overwhelming number of alerts every day. It is time consuming and error prone for engineers to investigate each alert manually without any guidelines. In this paper, we novelly propose a ranking-based framework named *AlertRank* that identifies severe alerts automatically and adaptively. One key component of *AlertRank* is a set of powerful and interpretable features to characterize the severities of alerts. These features include textural and temporal features from alerts, and univariate and multivariate anomaly features from monitoring metrics. We evaluate *AlertRank* using real-world data from a top global bank, and the results show that *AlertRank* is effective and achieves a F1-score of 0.89 on average, and significantly saves efforts devoted to investigate alerts for engineers.

Acknowledgement

We thank Junjie Chen for his helpful discussions on this work. We thank Juexing Liao, Yongqian Sun and Chuanxi Zheng for proofreading this paper. This work has been supported by the Beijing National Research Center for Information Science and Technology (BNRist) key projects and National Key R&D Program of China 2019YFB1802504.

REFERENCES

[1] Q. Lin, H. Zhang, J.-G. Lou, Y. Zhang, and X. Chen, "Log clustering based problem identification for online service systems," in *Proceedings of the 38th International Conference on Software Engineering Companion*, pp. 102–111, ACM, 2016.

[2] H. Xu, W. Chen, N. Zhao, Z. Li, J. Bu, Z. Li, Y. Liu, Y. Zhao, D. Pei, and et.al, "Unsupervised anomaly detection via variational auto-encoder for seasonal kpis in web applications," in *WWW*, 2018.

[3] V. Nair, A. Raul, S. Khanduja, V. Bahirwani, Q. Shao, S. Sellamanickam, S. Keerthi, S. Herbert, and S. Dhulipalla, "Learning a hierarchical monitoring system for detecting and diagnosing service issues," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 2029–2038, ACM, 2015.

[4] J. Xu, Y. Wang, P. Chen, and P. Wang, "Lightweight and adaptive service api performance monitoring in highly dynamic cloud environment," in *2017 IEEE International Conference on Services Computing (SCC)*, pp. 35–43, IEEE, 2017.

[5] G. Jiang, H. Chen, K. Yoshihira, and A. Saxena, "Ranking the importance of alerts for problem determination in large computer systems," *Cluster Computing*, vol. 14, no. 3, pp. 213–227, 2011.

[6] L. Tang, T. Li, F. Pinel, L. Shwartz, and G. Grabarnik, "Optimizing system monitoring configurations for non-actionable alerts," in *2012 IEEE Network Operations and Management Symposium*, pp. 34–42, IEEE, 2012.

[7] W. Zhou, W. Xue, R. Baral, Q. Wang, C. Zeng, T. Li, J. Xu, Z. Liu, L. Shwartz, and G. Ya Grabarnik, "Star: A system for ticket analysis and resolution," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 2181–2190, ACM, 2017.

[8] W. U. Hassan, S. Guo, D. Li, Z. Chen, K. Jee, Z. Li, and A. Bates, "Nodoze: Combatting threat alert fatigue with automated provenance triage.," in *NDSS*, 2019.

[9] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pp. 785–794, ACM, 2016.

[10] X. Yan, J. Guo, Y. Lan, and X. Cheng, "A biterm topic model for short texts," in *Proceedings of the 22nd international conference on World Wide Web*, pp. 1445–1456, ACM, 2013.

[11] C. Manning, P. Raghavan, and H. Schütze, "Introduction to information retrieval," *Natural Language Engineering*, vol. 16, no. 1, pp. 100–103, 2010.

[12] A. De Cheveigné and H. Kawahara, "Yin, a fundamental frequency estimator for speech and music," *The Journal of the Acoustical Society of America*, vol. 111, no. 4, pp. 1917–1930, 2002.

[13] S. Khatuya, N. Ganguly, J. Basak, M. Bharde, and B. Mitra, "Adele: Anomaly detection from event log empiricism," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pp. 2114–2122, IEEE, 2018.

[14] K. Hundman, V. Constantinou, C. Laporte, I. Colwell, and T. Soderstrom, "Detecting spacecraft anomalies using lstms and nonparametric dynamic thresholding," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 387–395, ACM, 2018.

[15] Q. Lin, K. Hsieh, Y. Dang, H. Zhang, K. Sui, Y. Xu, J.-G. Lou, C. Li, Y. Wu, R. Yao, *et al.*, "Predicting node failure in cloud service systems," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 480–490, ACM, 2018.

[16] N. Zhao, J. Zhu, R. Liu, D. Liu, M. Zhang, and D. Pei, "Label-less: A semi-automatic labelling tool for kpi anomalies," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pp. 1882–1890, IEEE, 2019.

[17] J. Chen, X. He, Q. Lin, Y. Xu, H. Zhang, D. Hao, F. Gao, Z. Xu, Y. Dang, and D. Zhang, "An empirical investigation of incident triage for online

[18] J. Chen, X. He, Q. Lin, H. Zhang, D. Hao, F. Gao, Z. Xu, D. Yingnong, and D. Zhang, "Continuous incident triage for large-scale online service systems," in *2019 34st IEEE/ACM International Conference on Automated Software Engineering (ASE)*, p. to appear, IEEE, 2019.

service systems," in *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice*, pp. 111–120, IEEE Press, 2019.

[19] J. MacQueen *et al.*, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, 1967.

[20] P. J. Rousseeuw, "Silhouettes: a graphical aid to the interpretation and validation of cluster analysis," *Journal of computational and applied mathematics*, vol. 20, pp. 53–65, 1987.

[21] S. Zhang, W. Meng, J. Bu, S. Yang, Y. Liu, D. Pei, J. Xu, Y. Chen, H. Dong, X. Qu, *et al.*, "Syslog processing for switch failure diagnosis and prediction in datacenter networks," in *2017 IEEE/ACM 25th International Symposium on Quality of Service (IWQoS)*, pp. 1–10, IEEE, 2017.

[22] C. C. Aggarwal and C. Zhai, *Mining text data*. Springer Science & Business Media, 2012.

[23] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *Journal of machine Learning research*, vol. 3, no. Jan, pp. 993–1022, 2003.

[24] M. Röder, A. Both, and A. Hinneburg, "Exploring the space of topic coherence measures," in *Proceedings of the eighth ACM international conference on Web search and data mining*, pp. 399–408, ACM, 2015.

[25] M. Vlachos, P. Yu, and V. Castelli, "On periodicity detection and structural periodic similarity," in *Proceedings of the 2005 SIAM International Conference on Data Mining, SDM 2005*, 04 2005.

[26] N. Zhao, J. Zhu, Y. Wang, M. Ma, W. Zhang, D. Liu, M. Zhang, and D. Pei, "Automatic and generic periodicity adaptation for kpi anomaly detection," *IEEE Transactions on Network and Service Management*, 2019.

[27] D. Liu, Y. Zhao, H. Xu, Y. Sun, D. Pei, J. Luo, X. Jing, and M. Feng, "Opprentice: Towards practical and automatic anomaly detection through machine learning," in *Proceedings of the 2015 Internet Measurement Conference*, pp. 211–224, ACM, 2015.

[28] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[29] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, p. 436, 2015.

[30] T.-Y. Liu *et al.*, "Learning to rank for information retrieval," *Foundations and Trends in Information Retrieval*, vol. 3, no. 3, pp. 225–331, 2009.

[31] C. J. Burges, "From ranknet to lambdarank to lambdamart: An overview," *Learning*, vol. 11, no. 23-581, p. 81, 2010.

[32] T. Zhang, J. Chen, G. Yang, B. Lee, and X. Luo, "Towards more accurate severity prediction and fixer recommendation of software bugs," *Journal of Systems and Software*, vol. 117, pp. 166–184, 2016.

[33] C. M. Bishop, *Pattern recognition and machine learning*. springer, 2006.

[34] D. Man, W. Yang, W. Wang, and S. Xuan, "An alert aggregation algorithm based on iterative self-organization," *Procedia Engineering*, vol. 29, pp. 3033–3038, 2012.

[35] S. Salah, G. Maciá-Fernández, and J. E. DíAz-Verdejo, "A model-based survey of alert correlation techniques," *Computer Networks*, vol. 57, no. 5, pp. 1289–1317, 2013.

[36] S. A. Mirheidari, S. Arshad, and R. Jalili, "Alert correlation algorithms: A survey and taxonomy," in *Cyberspace Safety and Security*, pp. 183–197, Springer, 2013.

[37] D. Lin, R. Raghu, V. Ramamurthy, J. Yu, R. Radhakrishnan, and J. Fernandez, "Unveiling clusters of events for alert and incident management in large-scale enterprise it," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 1630–1639, ACM, 2014.

[38] K. Alsubhi, E. Al-Shaer, and R. Boutaba, "Alert prioritization in intrusion detection systems," in *NOMS 2008-2008 IEEE Network Operations and Management Symposium*, pp. 33–40, IEEE, 2008.

[39] Y. Lin, Z. Chen, C. Cao, L.-A. Tang, K. Zhang, W. Cheng, and Z. Li, "Collaborative alert ranking for anomaly detection," in *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, pp. 1987–1995, ACM, 2018.

[40] Q. Umer, H. Liu, and Y. Sultan, "Emotion based automated priority prediction for bug reports," *IEEE Access*, vol. 6, pp. 35743–35752, 2018.