Real-Time Incident Prediction for Online Service Systems

Nengwen Zhao*
Tsinghua University; BNRist
Beijing, China

Junjie Chen†
College of Intelligence and Computing, Tianjin University
Tianjin, China

Zhou Wang
BizSeer; Beijing University of Posts and Telecommunications
Beijing, China

Xiao Peng
Gang Wang
China EverBright Bank
Beijing, China

Yong Wu
Fang Zhou
Zhen Feng
China EverBright Bank
Beijing, China

Xiaohui Nie
Tsinghua University; BNRist
Beijing, China

Wenchi Zhang
BizSeer
Beijing, China

Kaixin Sui
BizSeer
Beijing, China

Dan Pei
Tsinghua University; BNRist
Beijing, China

ABSTRACT

Incidents in online service systems could dramatically degrade system availability and destroy user experience. To guarantee service quality and reduce economic loss, it is essential to predict the occurrence of incidents in advance so that engineers can take some proactive actions to prevent them. In this work, we propose an effective and interpretable incident prediction approach, called *eWarn*, which utilizes historical data to forecast whether an incident will happen in the near future based on alert data in real time. More specifically, *eWarn* first extracts a set of effective features (including textual features and statistical features) to represent omen alert patterns via careful feature engineering. To reduce the influence of noisy alerts (that are not relevant to the occurrence of incidents), *eWarn* then incorporates the multi-instance learning formulation. Finally, *eWarn* builds a classification model via machine learning and generates an interpretable report about the prediction result via a state-of-the-art explanation technique (i.e., LIME). In this way, an early warning signal along with its interpretable report can be sent to engineers to facilitate their understanding and handling for the incoming incident. An extensive study on 11 real-world online service systems from a large commercial bank demonstrates the effectiveness of *eWarn*, outperforming state-of-the-art alert-based incident prediction approaches and the practice of incident prediction with alerts. In particular, we have applied *eWarn* to two large commercial banks in practice and shared some success stories and lessons learned from real deployment.

CCS CONCEPTS

• **Software and its engineering → Maintaining software**.

KEYWORDS

Incident Prediction, Online Service Systems, Real-time Prediction

ACM Reference Format:
Nengwen Zhao, Junjie Chen, Zhou Wang, Xiao Peng, Gang Wang, Yong Wu, Fang Zhou, Zhen Feng, Xiaohui Nie, Wenchi Zhang, Kaixin Sui, and Dan Pei. 2020. Real-Time Incident Prediction for Online Service Systems. In *Proceedings of the 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '20), November 8–13, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3368089.3409672

1 INTRODUCTION

Nowadays, online service systems, such as online shopping, E-bank, and search engines, have become an indispensable part in our daily life. Although tremendous efforts have been devoted to software service maintenance (e.g., collecting various monitoring data for a service system such as metrics [44, 46, 54], logs [19, 31, 51], traces [55], and alerts [29]), due to their large scale and complexity, incidents (i.e., unplanned interruption/outage to a service [2, 16, 25]) are still inevitable, which could lead to system unavailability and huge economic loss [32]. For example, according to a recent survey [1], the average cost per hour of server downtime is between $301,000 and $400,000.

To reduce the influence of incidents and guarantee the quality of software services, there are two widely-used ways in both academia and industry [32, 33], i.e., predicting the occurrence of an incident in advance so that engineers can take some proactive actions to prevent it [18, 43] and mitigate the already happened incident as soon as possible [14, 15]. Our work focuses on the first way since this way is able to directly avoid the occurrence of service unavailability rather than reduce the time of service unavailability.

*BNRist: Beijing National Research Center for Information Science and Technology
†Junjie Chen is the corresponding author.

ESEC/FSE '20, November 8–13, 2020, Virtual Event, USA

ACM ISBN 978-1-4503-7043-1/20/11…$15.00
https://doi.org/10.1145/3368089.3409672

In the literature, many efforts have been devoted to the first way, i.e., predicting the occurrence of incidents in advance [18, 41, 42, 48], but they still suffer from various limitations in practice. First, the vast majority of prediction approaches target at the prediction of a specific type of failures (such as disk failures [48], node failures [30], switch failures [50], and equipment failures [42]), and thus are restricting in practice. Moreover, these prediction approaches utilize logs or metrics to extract omen patterns for predicting the specific type of failures. However, tens of TBs of logs and more than thousands of metrics tend to be generated per day for a large-scale system [31], which could bring great challenges and costs to learn a prediction model from massive data. Second, AirAlert was proposed recently to predict general incidents and relies on lightweight alert data [18], but its performance is unsatisfactory since it just considers the number of different types of alerts for prediction, which has been demonstrated to be ineffective in our study (to be presented in § 4.2). Therefore, designing an effective and lightweight prediction approach for general incidents is very essential.

In this paper, we aim to propose a novel approach to predicting general incidents in real time. Similar to AirAlert, we also utilize lightweight alert data for prediction since alerts are more high-level and comprehensive. More specifically, alerts are generated to report anomalies from other monitoring data (e.g., metrics [46], logs [19, 31], and traces [55]), and thus avoid processing massive logs or metrics. However, predicting incidents based on alert data in practice also faces several challenges as follows. 1) Practical alert data contain tens of attributes (to be introduced in § 2.1), and thus how to extract useful information from them is challenging. 2) Not all alerts before the occurrence of an incident are helpful for prediction, and the existence of noisy alerts could destroy the prediction performance, and thus how to reduce the influence of noisy alerts is the second challenge. 3) In addition to accurately predicting the occurrence of an incident, an interpretable prediction result should be provided to engineers in order to facilitate them to understand and handle this incident, which is also challenging.

To overcome these challenges, in this paper we propose ***eWarn*** (short for **e**arly **Warn**ing), an approach utilizing historical data to predicting incidents in real time based on alert data. In particular, we formulate the problem of incident prediction as a binary classification task of observation windows (to be presented in § 2.3), where positive and negative samples refer to whether an incident will occur or not within a particular time horizon based on the alert data within the corresponding observation window respectively. In detail, *eWarn* first conducts feature engineering to extract a set of effective and interpretable features from alert data within an observation window. Then, *eWarn* incorporates multi-instance learning [11, 42] to reduce the influence of noisy alerts by assigning smaller weights to noisy alerts. Next, *eWarn* processes the data imbalance problem due to the small frequency of incidents in practice and builds a classification model via machine learning (XGBoost [17]). After getting the prediction result in real time by the learned model, an early warning would be sent to engineers if an incident is predicted to occur in the near future. Along with the prediction result, *eWarn* also utilizes LIME [38], a state-of-the-art explanation technique, to generate a report for engineers to interpret the prediction result in a visualization manner.

We conducted an extensive study to investigate the performance of *eWarn* based on 11 real-world online service systems in a large commercial bank (named $\mathcal{A}$ in this paper due to the double-blind policy), which supports more than one hundred million users and includes hundreds of service systems. For each system, engineers provided us three-year alert data and incidents. Our experimental results show that *eWarn* performs the best by comparing with the state-of-the-art alert-based incident prediction approach (i.e., AirAlert [18]), a novel approach to predicting a specific type of failures based on logs [49] (we adapted it to fit our problem, i.e., general incident prediction based on alerts), and the current practice of incident prediction with alerts in bank $\mathcal{A}$. For example, the average F1-score of *eWarn* is 0.82 while those of the other three approaches are 0.51, 0.60, and 0.10 respectively. Also, our results confirm the contributions of main components in *eWarn* (i.e., the multi-instance learning component, the feature engineering component, and the classification model building component) to the overall performance of *eWarn*. In particular, we have applied *eWarn* to two large commercial banks (including $\mathcal{A}$) and indeed achieved great effectiveness in practice. We have presented four specific cases in practice and discussed lessons learned in § 5.

To sum up, this work has the following major contributions:

- We propose an effective and interpretable approach, called *eWarn*, to predicting general incidents in real time based on alert data, which could send an early warning signal about an incoming incident including an interpretable prediction result for engineers to facilitate them to adopt proactive actions to avoid the incident in advance.
- We conduct an extensive study based on 11 real-world online service systems (engineers provided us three-year alert data and incidents for each system) by comparing state-of-the-art incident prediction approaches and the current practice for incident prediction based on alert data. Our experimental results demonstrate the effectiveness of *eWarn* and confirm the contributions of main components in *eWarn*.
- We have applied *eWarn* to two large commercial banks, both of which support more than one hundred million users, and indeed achieved great effectiveness. We also shared four specific cases during the practical usage of *eWarn*.

2 MOTIVATION & PROBLEM FORMULATION

2.1 Background: Alert and Its Management

For online service systems, alerts are a key data source for recording the anomalies generated from various system components. More specifically, monitoring systems continuously collect various data (e.g., metrics [46], logs [19, 31], and traces [55, 56]) from various service components, and engineers manually define many rules to check these monitoring data to ensure service availability. When a certain rule is violated, an alert would be generated to report the anomaly. For example, when a metric (e.g., CPU utilization) exceeds a pre-defined threshold or some keywords (e.g., "error", "failed" or "timeout") appear in logs, alerts would be generated. Table 1 shows examples of alerts from EPAY service in bank $\mathcal{A}$. Each alert has many attributes, and we only show several important attributes due to the space limit. In this table, the columns represent the occurring time of an alert, the detailed textual description about an alert, the

Table 1: Examples of alerts with multiple attributes

Time	Content	Server	Service	Severity	Type	Others
2020-02-03 08:24:11	Authentication failure for SNMP request from host P13.	P10	EPAY	3	Network	...
2020-02-03 08:25:34	Can't get Weblogic queue (EPAYAPP). Timeout.	P31	EPAY	2	Middleware	...
2020-02-03 08:26:04	The utilization of file system /home/etl441 is 82%, exceeding 80%.	P72	EPAY	2	OS	...
2020-02-03 08:26:51	Business success rate is 88%, lower than 90%.	P2	EPAY	1	Application	...

server generating an alert[1], the service generating an alert, the severity of an alert ("1" refers to the highest severity), the type of an alert, respectively.

Due to the importance of alerts, an alert management system is important for service system maintenance. Usually, when alerts are sent to the alert management system, the system handles them by three main steps: 1) Alert pre-processing, which filters out duplicate and redundant alerts by the means of alert denoising, compression, and/or correlation [29, 35, 47]. 2) Alert diagnosis, which prioritizes alerts [24], and then assigns each alert to the responsible team and engineers in the team diagnose it as soon as possible [14]. 3) Alert post-processing, which writes diagnosis reports and gains insights to guide the future improvement of alert management.

2.2 Practice of Incident Prediction with Alerts

By communicating with our industry partners, i.e., two top commercial banks $\mathcal{A}$ and $\mathcal{B}$ in the country, both of them confirmed the importance and feasibility of incident prediction based on alerts so that some proactive actions can be adopted in advance to avoid incidents and the corresponding economic loss. Moreover, both of them have tried to adopt some simple methods to predict incidents based on alerts in practice.

The practice of incident prediction with alerts in $\mathcal{A}$ is that, they use the technique of association rule mining [21] (e.g., FP-Growth [22]) to automatically discover omen alerts for incidents. For example, if an alert always happens before an incident, this alert would be used to predict the incident. However, according to feedback from engineers, this method can only cover a very small set of incidents since most incidents do not have frequently associated alerts and cannot be accurately predicted in this way.

The practice of incident prediction with alerts in $\mathcal{B}$ is that, they devote many efforts to manually summarize a set of prediction rules based on their experience and domain knowledge. Table 2 presents some examples of rules summarized for incident prediction in $\mathcal{B}$. Taking the first one as an example (No.1), if the alerts with the keyword "TCP is not responding" appear at least once (#Matching), last for at least three minutes (Duration), involve four different servers (#Servers), and do not happen in the period of software change (Change), engineers infer that the involved servers would be down (Incident). However, the rule-based incident prediction method performs not well in practice due to the following reasons. First, it is time-consuming and tedious to manually summarize these rules from a large amount of historical data. Second, the design and maintenance of the rules require experienced experts with rich domain knowledge, and in the meanwhile different engineers tend to have their own preference when designing rules. Third, the rule-based method cannot adapt to the dynamic environment and concept drift [45] can largely affect manually-summarized rules, causing that engineers have to spend many efforts in manually checking and re-summarizing the rules.

In summary, it is indeed important to predict incidents based on alert data but the current practices are very unsatisfactory. Thus, it is necessary to propose an effective automatic incident prediction approach based on alert data, largely motivating our work.

[1]Due to confidentiality, we do not disclose the IP addresses and use P* to denote different servers.

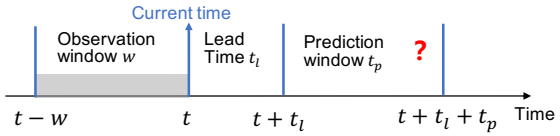

Figure 1: Problem formulation of incident prediction

2.3 Problem Formulation

Motivated by existing related works [37, 40, 50], we formulate the problem of incident prediction as *time window classification*. We use Figure 1 to illustrate the problem formulation. To ensure the real-time online prediction, we adopt the strategy of moving time window with size w and a fixed step Δt. For the current time t, we get the *observation window* $[t-w, t]$. Then, we use alert data within the observation window to predict whether an incident will occur within the *prediction window* $[t+t_l, t+t_l+t_p]$. *Lead time* (t_l) in Figure 1 is the minimum time interval that engineers need to react to an early warning (e.g., master-standby switch). That is, we have to leave enough time (i.e., Lead time) for engineers to handle the early warning after predicting an incident. To complete the prediction task, collecting training data is required. In historical data, if there is an incident occurring within the prediction window, the corresponding observation window is labeled as a positive sample (i.e., an omen window). Otherwise, it is labeled as a negative sample (i.e., a non-omen window).

Note that we adopt the moving window strategy to ensure real-time prediction, and thus an incident could appear in several prediction windows (since Δt tends to be smaller than t_p, some prediction windows have overlaps). That is, we could obtain several adjacent positive windows (omen windows) for an incident, and the number of positive windows is $t_p/\Delta t$. The larger/smaller the size of a prediction window is, the more/less positive windows we can obtain. That is, the value of t_p could directly affect the labeling quality, which will be discussed in detail in § 4.5. Other parameter settings (i.e., Δt, w, and t_l) will be presented in detail in § 4.1.3.

3 APPROACH

In this section, we propose a real-time and lightweight approach, named ***eWarn*** (short for **e**arly **Warn**ing), to predicting incidents

Table 2: Illustrating examples of rule-based incident prediction with alerts

No.	Keywords	#Matching	Severity	Duration	#Servers	Change	Incident
1	TCP is not responding	1	2	3min	4	No	Server may be down
2	CPU & currently 100%	2	2	5min	2	No	System performance degradation
3	#Sessions exceeds threshold	2	2	5min	1	No	Transaction with long response time

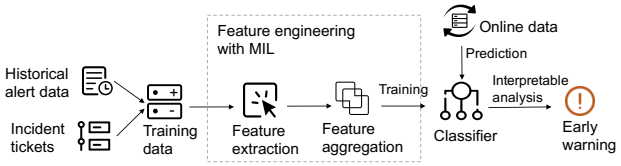

Figure 2: Overview of *eWarn*

in advance based on alert data. According to § 2.3, for a system, we collect a set of time windows (i.e., observation windows) with historical alert data and their labels as training data. However, to solve the problem, *eWarn* has to overcome the following challenges: 1) Since an alert contains many attributes presented in § 2.1, which alert information should be identified as the features for effectively predicting incidents is the first challenge; 2) Since an observation window contains a number of alerts and not all the alerts are helpful to anticipate incidents, how to reduce the influence of noisy alerts is the second challenge; 3) After predicting an incident, it is also important to interpret the relationship between the predicted incident and the alerts in the observation window, which is helpful for engineers to understand and handle this incident in practice and is also the third challenge.

Figure 2 shows the overview of *eWarn*. *eWarn* consists of four main steps. First, *eWarn* conducts feature engineering to identify a set of effective and interpretable features from alert data in an observation window to overcome the first challenge. Second, *eWarn* incorporates multi-instance learning (MIL) [11, 42] to distinguish helpful alerts and noisy alerts in an observation window to overcome the second challenge. Third, based on the set of training data with identified features and labels, *eWarn* further processes them (i.e., dealing with class imbalance) and builds a classification model via machine learning, which is used to predict whether an incident will occur within a prediction window based on incoming alert data in real time. Fourth, after predicting an incident, *eWarn* sends an early warning to notify engineers. In the meanwhile, *eWarn* adopts LIME [38], a state-of-the-art explanation technique, to interpret the prediction results for engineers to facilitate the understanding and handling of the incident, which is used to overcome the third challenge. In the following, we present each step in detail.

3.1 Identifying Features

We identify two types of features from alert data to predict incidents in *eWarn*, i.e., textual features and statistical features [53].

3.1.1 Textual Features. As shown in Table 1, the most informative attribute in alert data is its textual content, and the rule-based incident prediction method also focuses on the alert content (Keywords in Table 2). Therefore, we extract textual features from alert contents as the first type of features in *eWarn*.

In the literature, many existing methods have been proposed for textual feature extraction or text representation, which can be divided into two categories. The first one is statistical method, such as Term Frequency-Inverse Document Frequency (TF-IDF) [8] and Topic Model [10], which are based on word frequency. The second one is neural network based method, such as word2vec [34], TextCNN [27], and FastText [26]. Here, we choose *Topic Model* in the first category to extract textual features in *eWarn* due to the following reasons. First, the neural network based methods depend on a huge amount of training data and suffer from high computation costs. Moreover, it is difficult to gain interpretable insights from black-box models built via neural networks. Second, the classical TF-IDF method could produce a very high-dimensional feature vector, leading to the curse of dimensionality problem in machine learning.

In particular, *eWarn* adopts the widely-used topic model, Latent Dirichlet Allocation (LDA) [10], to extract textual features. In LDA, each document (referring to all the alert contents within a time window in *eWarn*) is viewed as a mixture of various topics with different probabilities and the topic probability distribution is used as the feature vector of the time window. Each topic is characterized by a distribution of words that frequently co-occur in the document. Hence, LDA can find hidden semantic information existing in the window, which is helpful for incident prediction since positive windows may have similar topic distributions. More specifically, *eWarn* first treats all the alert contents within a time window as a document by concatenating these contents after stop words removal and tokenization, then uses historical time windows to build an LDA model, and finally obtains feature vectors based on the LDA model for both incoming time windows and historical time windows.

Note that LDA requires an input, i.e., the number of topics (denoted as K). The setting of K should consider the trade-off between more coarse-grained (smaller K) and more fine-grained topics (larger K). *eWarn* determines the number of topics based on the *coherence score* [39], which is a popular metric to evaluate the quality of output topics and the corresponding keywords. As a result, after feature representation via LDA, *eWarn* obtains the textual feature vector with K dimensions for each time window, where each element refers to the probability of each topic.

3.1.2 Statistical Feature. In addition to textual features extracted from alert contents, inspired by Table 2, some other attributes are also important for incident prediction and we identify them as the second type of features in *eWarn*, called statistical features [53]. In particular, we summarize the extracted statistical features in *eWarn* into the following four categories:

- *Alert count*, refers to the number of alerts that occur within a time window, including the total number of alerts, the number of alerts with different severities ($1 \sim 3$), the number of alerts with different types (e.g., application, database, memory, middleware, network, hardware, etc.). We infer that the more alerts there are in the window, the more likely incidents will occur.

- *Window time*, refers to the time of a window. Considering the occurrence of incidents may be related to time (e.g., some incidents tend to occur during business hours), we collect a series of time features, including hour of the day, weekend or not, day of week, business hours or not, etc.
- *Inter-arrival time*, refers to the average time interval between contiguous alerts in a time window. Intuitively, if alerts occur more intensively, an incident is more likely to occur.
- *Others*. Inspired by Table 2 and domain knowledge, we also consider additional features such as the number of alerting servers during a time window (the more servers involved, the more likely an incident will occur), whether in the software change period (software change is error-prone and needs more attention).

In summary, all these features used in *eWarn* are identified based on careful data analysis and domain knowledge from experts. Discussions with several industry partners demonstrate the generality of these features in *eWarn*. Furthermore, *eWarn* also supports adding or removing some features according to the application scenarios, but the core idea and pipeline of *eWarn* are general. In particular, we also conduct an experiment to investigate the performance of each type of feature, which will be presented in § 4.3.2.

3.2 Bypassing Noisy Alerts via Multi-Instance Learning

Usually, features are directly extracted from each observation window. However, not all alerts in an observation window are helpful to predict incidents, and the omen alerts may be flooded in the non-omen alerts. Due to the existence of noisy alerts in an observation window, directly extracting features from the observation window could lead to performance degradation [42, 50].

To reduce the influence of noisy alerts, *eWarn* adopts multi-instance learning (MIL) [11]. It splits an observation window into multiple small instance windows, where an observation window is regarded as a bag and each bag contains multiple instances (i.e., instance windows). MIL explores more fine-grained time windows (i.e., instance windows with size t_i) to bypass noisy alerts. That is, under the framework of MIL, *eWarn* first extracts features from each instance window instead of extracting features from the whole observation window, and then aggregates the features of these instances into the features of a bag that is used for building a classification model. If an instance window does not contain many helpful alerts, it is feasible to assign a small weight to the instance during the aggregation process and thus reduce the influence of noisy alerts. In the following we present the brief background of MIL and how to aggregate instance features to a bag feature.

Brief Background of MIL. MIL assumes that negative bags contain only negative instances and positive bags contain at least one positive instance. Formally, let Y be the label of a bag X containing a set of instances $X = \{x_1, x_2, \cdots, x_m\}$. Each instance x_i corresponds to a label y_i. The label of the bag is given by:

$$Y = \begin{cases} +1, & \text{if } \exists_{y_i} \quad y_i = +1, \\ -1, & \text{if } \forall_{y_i} \quad y_i = -1. \end{cases} \tag{1}$$

For incident prediction, it is also more reasonable to assume that at least one instance before the occurrence of an incident carries an omen signature than to assume that all the instances in the observation window carry an omen signature, which is another reason why MIL is helpful to reduce the influence of noisy alerts.

Clustering-Based Feature Aggregation. There are some existing works about bag feature aggregation in the field of MIL, e.g., simple feature averaging (or using minimum/maximum feature) [42], using neural network to learn instance weights [36], and adopting some statistical methods to compute instance weights [12].

In *eWarn*, it directly uses feature averaging to obtain bag features for negative bags, since all the instances in a negative bag are non-omen and they would share the same weight. For positive bags, the instances that contain many omen alerts (omen instances) should be assigned to larger weights, while the instances that contain a lot of noisy alerts (non-omen instances) should be assigned to smaller wights. Here, *eWarn* distinguishes omen alerts and noisy alerts by the method of clustering based on the observation that omen instances tend to be similar and appear more than once in training data, while non-omen instances tend to be various and even chaotic.

More specifically, *eWarn* adopts Hierarchical Clustering [20] to cluster the instances from all the positive bags in training data. Actually, *eWarn* is not specific to this clustering algorithm, and we choose it since it does not require to pre-define the number of expected groups. With our above-mentioned observation, we suppose the clusters with more instances are more likely to be the clusters of omen instances and thus the instances in these clusters should get larger weights, while the clusters with few instances (also called outliers in clustering) are more likely to be the clusters of non-omen instances. Formally, for the bag $X = \{x_1, x_2, \cdots, x_m\}$, the weight of the instance x_i ($i = 1, \ldots, m$) is calculated as:

$$w_i = \frac{w_i'}{\sum_{j=1}^{m} w_j'} \quad , \quad w_i' = \frac{n(c_{x_i})}{n} \tag{2}$$

where the left formula aims to normalize the weight w_i', c_{x_i} is the cluster that x_i belongs to, $n(c_{x_i})$ is the size of the cluster c_{x_i}, m is the number of instances in this bag, and n is the total number of instances from all the positive bags in training data. For the instance windows in testing data with unknown bag labels, we assign each instance window to the corresponding cluster based on the distance between the instance to each cluster center, and the weight can also be obtained using Eq.(2).

After obtaining the weight of each instance in a bag, the aggregated feature for this bag is calculated as:

$$f_b = \sum_{i=1}^{m} (w_i \times f_i) \tag{3}$$

where f_i is the feature of the instance x_i. We also conduct an experiment to investigate the necessity and effectiveness of MIL in *eWarn*, which will be presented in § 4.3.1.

3.3 Building a Classification Model

After obtaining bag features, *eWarn* will build a classification model via machine learning. Before that, it first addresses the problem of data imbalance. More specifically, for an online service system, the occurring frequency of incidents is very low, otherwise the service quality and user experience would be destroyed. Therefore, the number of positive observation windows and negative ones are very imbalanced, which leads to a significant challenge to the

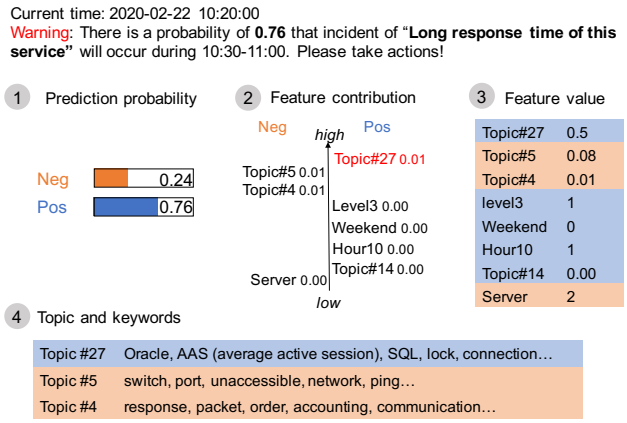

Figure 3: An example interpretable report on a prediction

objective of simultaneously achieving both high precision and high recall for machine learning based incident prediction.

To better handle imbalance data, *eWarn* adopts the widely-used SMOTE [13] oversampling strategy to balance the positive samples and negative ones, in which the minority class (positive windows) is over-sampled by creating synthetic examples through finding k-nearest neighbors along the minority class. In particular, we also conduct an experiment to show the effectiveness and necessity of adopting SMOTE in *eWarn*, which will be presented in § 4.3.3.

Based on the processed training set, *eWarn* adopts the gradient boosting tree based model (XGBoost) [17] to build a classification model. XGBoost is fundamentally a regression tree that has the same decision rule as the traditional decision tree model. Some of the major benefits of XGBoost are its high scalability/parallelism, efficiency, and outstanding performance. Besides, tree-based models naturally have the ability of feature selection. With the built classification model, in the running process of the system, *eWarn* predicts whether an incident will occur within the prediction window in real time based on the current observation window.

3.4 Interpretability Analysis

When an incident is predicted by *eWarn*, an early warning will be sent to notify engineers, so that they can adopt some proactive actions in advance. In addition, *eWarn* also provides an explanation report about the prediction along with the early warning, to facilitate engineers to understand and handle this incident in practice. Here, *eWarn* adopts Local Interpretable Model-agnostic Explanations (LIME) [38] to explain a prediction result by providing relative feature contributions for a single sample to the prediction result. More specifically, LIME is developed to identify an interpretable model that is locally faithful for each individual prediction. It learns a locally weighted linear model on this neighborhood data to explain each of classes in an interpretable way.

We illustrate an example report for engineers generated by *eWarn* in Figure 3. The generated report contains four parts. The first part reports the predicted probability that an incident will occur within the prediction window based on an observation window. The second part reports feature contributions, whose values are the weights calculated by the linear model in LIME (approximating the behavior of our XGBoost classifier). The value of each feature is presented in the third part ranking based on feature contributions. The report also contains detailed information (corresponding keywords) of each topic (the fourth part), so that engineers can understand each topic intuitively. In particular, the most important feature may be related to the root cause of the predicted incident, which may assist engineers in incident troubleshooting. For example, the most important feature in this example is Topic#27, whose keywords are all related to database, and thus we infer that the root cause comes from database (we will discuss the real case in detail in § 5.1).

Therefore, with an explainable report, engineers not only benefit from early warnings of incidents, but also gain some inspiration for incident diagnosis.

Table 3: Statistical information of datasets

System	#Alerts	#Incidents	#Positive	#Negative
S1	18,821	173	524	8,460
S2	13,315	214	392	7,907
S3	14,211	59	322	4,014
S4	9,499	27	161	6,176
S5	9,592	48	165	7,886
S6	13,811	39	101	8,603
S7	6,766	46	272	3,310
S8	9,808	26	149	1,873
S9	8,770	72	510	6,196
S10	127,619	227	1,125	15,035
S11	69,999	148	1,012	13,057

4 EVALUATION

In the study, we aim to address the following research questions:

- RQ1: How does *eWarn* perform in predicting incidents?
- RQ2: Does each main component in *eWarn* contribute to *eWarn*?
- RQ3: How does *eWarn* perform in terms of efficiency?
- RQ4: How the choice of parameters affects *eWarn*?

4.1 Experimental Setup

4.1.1 Dataset. In the study, we used 11 representative online service systems from bank $\mathcal{A}$, which supports more than one hundred million users and includes hundreds of service systems (We do not disclose the service names due to the confidentiality in the bank.). For each system, engineers provided us three-year alert data and incidents. Table 3 presents the details of the datasets used in our study, including the number of alerts, the number of incidents, and the number of positive/negative windows for each online service system. Note that, if an observation window contains too few alerts (i.e., less than 3 alerts), it is virtually impossible to extract either the omen pattern or the non-omen pattern from it, and thus we discarded these windows in the study. We split these data into training, validation and testing sets with a ratio of 6: 2: 2 in the chronological order. Note that, the positive samples and negative samples are indeed imbalance, ranging from 1:10 to 1:100.

4.1.2 Measurements. In the study, we adopted the widely-used classification metrics, i.e., precision, recall, and F1-score, as our measurements. *Precision* ($\frac{TP}{TP+FP}$) measures the percentage of incidents really occurring within the prediction windows among all the predicted incidents. *Recall* ($\frac{TP}{TP+FN}$) measures the percentage

of incidents that are correctly predicted in advance among all the incidents. *F1-score* is the harmonic mean of precision and recall. In addition, a classification threshold is required to give a binary result from probability and compute precision/recall, and here we decided the threshold based on the best performance on the validation set. Actually, false negative is much severer than false positive for incident prediction, since the cost of missing an incident is much larger than that of investigating a false alarm. Therefore, we pay more attention to *recall* in our study.

Furthermore, considering the practicability of our proposed approach in the real world, we also measured the time efficiency of *eWarn*. We recorded the time spent on the offline training stage, which learns a classifier for observation window classification, and the time spent on the online prediction stage, which provides a prediction result for the current observation window in real time.

4.1.3 Parameters and Environment. There exist several parameters in the problem formulation of *eWarn*, i.e., observation window size (w), step of moving window (Δt), lead time (t_l), prediction window size (t_p) and instance size (t_i). Based on grid search of parameters and the discussions with our industry partners (i.e., banks $\mathcal{A}$ and $\mathcal{B}$), we set $\Delta t = 10min$ (*eWarn* conducts a prediction every 10 minutes), $w = 60min$, $t_l = 10min$ and $t_i = 10min$. The selection of prediction window size will be discussed in § 4.5.

About the parameters in *eWarn*, the number of topics is decided based on coherence score [39] (presented in § 3.1.1). The positive samples are oversampled by SMOTE with a 1:10 sampling rate. We used the default settings of XGBoost provided by the xgboost library [7]. For the compared approaches (to be introduced in § 4.2), we set the support threshold in FP-Growth to be 0.3 and used the parameter settings of AirAlert and TF-IDF-LSTM provided by their papers respectively [18, 49]. For the hyperparameters of TextCNN [27] (§ 4.3.2), we set them as follows: the word-embedding size is 100, the kernel size and stride of 1-D convolution layer are 8 and 1 respectively, the pool size and stride of the average pooling layer are 2 and 1 respectively, the hidden size of the fully connected layer is 128. For the hyperparameters of FastText [26] (§ 4.3.2), we set them as follows: the embedding model is cbow, the size of the context window is 5, and the size of word vectors is 100. For the compared Deep Neural Network method for building a classification model (§ 4.3.3), we used three fully connected layers and the hidden size is 128. All these three neural network based methods adopt the adam optimizer with the learning rate of 0.01 and 30 epochs. We set these hyperparameters based on the grid search and the validation set.

eWarn and all the compared approaches are implemented by Python with widely-used libraries, including NumPy [4],pandas [5], scikit-learn [6], xgboost [7] and Keras [3]. Our study was conducted on Ubuntu 18.04.1 with 24-core Intel Xeon(R) CPU E5-2620 v3 @ 2.40GHz, 64 GB memory, 64-bit operating system.

4.2 Overall Performance of *eWarn*

To answer RQ1, we compared *eWarn* with the following existing approaches to demonstrate the effectiveness of *eWarn* in incident prediction based on alert data.

- AirAlert [18], which is the state-of-the-art alert-based incident prediction approach, uses the number of different kinds of alerts within an observation window as features and utilizes XGBoost to conduct binary classification.
- TF-IDF-LSTM [49] is proposed to predict a specific type of failure (cluster failure) based on log data, which can be adapted to our problem (incident prediction based on alert data). It uses TF-IDF to extract textual features from the observation window and uses Long short-term memory (LSTM) [23] to predict incidents. Although some other recent works also focus on predicting a specific type of failures [28, 50], they are designed for a specific scenario and thus cannot be applied to alert data.
- FP-Growth [22] is a classical association rule mining method [21], and can be used to predict incidents by mining the correlated alerts that always occur before incidents occur. This approach is the practice of bank $\mathcal{A}$ presented in § 2.2. More specifically, it mines the frequent alerts with high support and confidence from historical data before the occurrence of incidents. During the process of online prediction, once these alerts occur, it predicts that an incident will occur within the prediction window.

Table 4 shows the precision, recall, and F1-score comparison results between *eWarn* and these compared approaches for 11 online service systems. From this table, we find that *eWarn* outperforms all the three compared approaches on average in terms of all the measurements (i.e., precision, recall, and F1-score). In particular, the best F1-score of *eWarn* achieves 0.98 and its average F1-score is 0.82, while the average F1-score of AirAlert, TF-IDF-LSTM, and FP-Growth is only 0.51, 0.60, and 0.10, respectively. Please note that, not the incidents of all systems can be predicted with very high precision and recall (e.g., S3 and S8). This is because some incidents may occur by chance due to some unexpected factors and thus can not be predicted in advance, which will be discussed in detail in § 5.2. Besides, considering false negative is more important than false positive in our problem, *eWarn* indeed achieves stably high recall compared with the three compared approaches in general. In particular, the worst recall of *eWarn* is 0.69, while the worst recall of AirAlert, TF-IDF-LSTM, and FP-Growth is 0.24, 0.31, and even 0, respectively. Actually, *eWarn* can achieve higher recall (e.g., 100% recall with no false negatives) by tuning the classification threshold, but this will also lead to precision degradation.

We further analyzed the reasons why the compared approaches perform not well. For AirAlert, it only utilizes the number of each kind of alert as features, which cannot represent the complex alert data very well. The hidden semantic features of alerts, alert time and some other information should also be considered. TF-IDF-LSTM extracts textual features by TF-IDF, which is based on only word frequency, and in the meanwhile TF-IDF could produce high-dimensional feature vectors, which is unfriendly for classification. Besides, the moving time window formulation in *eWarn* (presented in Section 2.3) has considered the time dependency, and thus LSTM cannot make additional contributions to the performance compared with *eWarn* and it also increases the computation costs (to be presented in § 4.4). For FP-growth, it performs rather poorly since omen patterns are very complicated and cannot be captured by simple alert matching. Besides, since an alert data has multiple attributes (shown in Table 1) and the alert content has many parameters (e.g., different metric values, ports, API names, IP addresses, etc.), it is very hard to find frequent alerts from historical positive

Table 4: Precision (P), recall (R) and F1-score (F) comparison between *eWarn* and compared approaches

Approach	*eWarn*			AirAlert			TF-IDF-LSTM			FP-Growth			W/o MIL		
System	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F
S1	0.86	0.82	**0.84**	0.46	0.82	0.59	0.93	0.73	0.82	0.08	0.05	0.06	0.36	0.80	0.50
S2	0.86	0.97	**0.91**	0.81	0.94	0.87	0.80	0.88	0.84	0.25	0.22	0.23	0.82	0.97	0.89
S3	0.61	0.83	**0.70**	0.41	0.24	0.31	0.23	0.76	0.35	0.05	0.09	0.07	0.50	0.67	0.57
S4	0.92	0.84	**0.88**	0.34	0.81	0.48	0.58	0.39	0.46	0.16	0.27	0.20	0.97	0.52	0.68
S5	0.75	0.86	**0.80**	0.34	0.29	0.32	0.14	0.31	0.19	0.12	0.25	0.17	0.71	0.39	0.51
S6	0.96	1.00	**0.98**	0.21	1.00	0.35	0.91	1.00	0.95	1.00	0.05	0.09	0.96	1.00	0.98
S7	0.73	0.71	**0.72**	0.65	0.53	0.59	0.67	0.73	0.69	0.00	0.00	0.00	0.36	0.76	0.49
S8	0.56	0.92	**0.69**	0.22	1.00	0.36	0.17	1.00	0.30	0.13	0.10	0.11	0.60	0.61	0.61
S9	0.92	0.98	**0.95**	0.53	1.00	0.69	0.92	0.98	0.95	0.03	0.02	0.02	0.91	0.98	0.95
S10	0.70	0.79	**0.76**	0.55	0.86	0.67	0.52	0.90	0.66	0.53	0.06	0.11	0.51	0.92	0.66
S11	0.81	0.69	**0.75**	0.28	0.57	0.37	0.25	0.52	0.34	0.01	0.06	0.01	0.41	0.53	0.46
Average	–	–	**0.82**	–	–	0.51	–	–	0.60	–	–	0.10	–	–	0.66

windows. More importantly, all of the three approaches do not deal with the negative effect caused by noisy alerts.

Overall, compared with the three approaches, *eWarn* is indeed able to predict incidents with higher precision, recall, and F1-score.

4.3 Contributions of Main Components

To answer RQ2, we investigated the contributions of three main components in *eWarn* to its overall performance, including the multi-instance learning formulation, the feature engineering component, and the classification model building component.

4.3.1 Contribution of Multi-instance Learning Formulation. As presented in § 3.2, *eWarn* adopts multi-instance learning [11, 42] to reduce the influence of noisy alerts. To investigate the contribution of MIL in *eWarn*, we compared *eWarn* with a variant of *eWarn* that directly extracts features from each observation window without splitting it into multiple instance windows. The last column (W/o MIL) in Table 4 shows the performance of the variant of *eWarn*. We find that the variant of *eWarn* without MIL indeed performs worse than *eWarn* in terms of all the measurements (i.e., precision, recall, F1-score) in general. In particular, the average F1-score drops from 0.82 to 0.66 after removing MIL from *eWarn*. In the meanwhile, the performance of the variant of *eWarn* is the same as that of *eWarn* for two online service systems (i.e., S6, and S9), probably because there are little noisy data for them. Overall, our multi-instance learning formulation is indeed effective to bypass noisy data and is able to improve the F1-score by nearly 20% on average.

4.3.2 Contribution of Feature Engineering. Faced with complex alert data with multiple attributes, *eWarn* extracts a set of powerful features to represent alert patterns, including textual features and statistical features. To demonstrate the contribution of our feature engineering, we first investigated the effectiveness of each type of feature used in *eWarn* (i.e., textual features and statistical features). That is, we evaluated the effectiveness of *eWarn* using only one type of feature, whose results are shown in Table 5 (the third and fourth columns). We find that *eWarn* incorporating both kinds of features achieves an average F1-score of 0.82 while the average F1-score of *eWarn* with only textual features is 0.69 and that of *eWarn* with only statistical features is only 0.36. The results demonstrate the contribution of each type of feature, and textual features are more powerful than statistical features for incident prediction.

Table 5: F1-score comparisons to demonstrate the effectiveness of feature engineering in *eWarn*

System	*eWarn*	Only Textual	Only statistical	TextCNN	FastText
S1	0.84	0.62	0.51	0.54	0.57
S2	0.91	0.88	0.19	0.34	0.40
S3	0.70	0.48	0.30	0.37	0.43
S4	0.88	0.73	0.26	0.45	0.47
S5	0.80	0.57	0.41	0.50	0.53
S6	0.98	0.90	0.38	0.61	0.65
S7	0.72	0.69	0.44	0.56	0.52
S8	0.69	0.48	0.37	0.38	0.41
S9	0.95	0.84	0.29	0.42	0.48
S10	0.76	0.70	0.49	0.64	0.69
S11	0.75	0.68	0.35	0.47	0.45
Average	0.82	0.69	0.36	0.48	0.51

We then investigated the effectiveness of our white-box interpretable method for textual feature representation (i.e., LDA) compared with two popular black-box neural network based textual feature representation methods, i.e., TextCNN [27] and FastText [26], whose results are shown in the last two columns in Table 5. We find that *eWarn* using LDA significantly outperforms the two neural network based methods. The average F1-score of the former is 0.82 while those of the latter two are only 0.48 and 0.51, respectively. This is because the two methods designing for natural language rely on a sufficiently large training corpus. However, strictly speaking, alert contents are domain-specific (IT operations) and semi-structured short texts, so the two methods perform not well in our scenario. More importantly, *eWarn* does not use the two methods since features extracted from these black-box models are hardly associated with domain knowledge directly. In contrast, *eWarn* carefully designs a set of effective textual features and statistical features based on the white-box interpretable method and domain experience, which not only have physical significance but also can be easily understood by engineers.

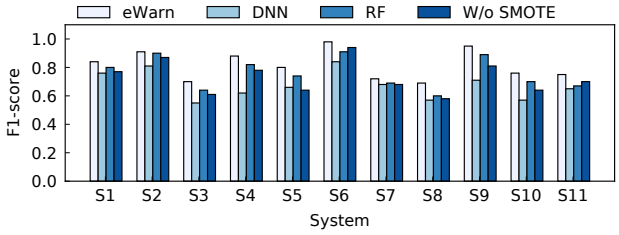

Figure 4: F1-score comparisons to demonstrate the effectiveness of the classification model building in *eWarn*

4.3.3 Effectiveness of Classification Model Building. We evaluated the effectiveness of the classification model building in *eWarn* from two aspects: building a classifier via XGBoost and handling the data imbalance problem.

To investigate the effectiveness of building a classifier via XGBoost in *eWarn*, we compared XGBoost with two widely-used machine learning algorithms, i.e., Deep Neural Network (DNN) [9] (the used network structure has been introduced in § 4.1.3) and Random Forest (RF) [9], whose results are shown in Figure 4. We find that XGBoost indeed outperforms the other two algorithms and RF also performs well in terms of F1-score, indicating that our features can be modeled well by tree structures. The results demonstrate the effectiveness of XGBoost for incident prediction.

To investigate the necessity of dealing with the data imbalance problem, we compared *eWarn* with a variant of *eWarn* that does not deal with this data imbalance problem (i.e., removing SMOTE from *eWarn*), whose results are shown in Figure 4 (W/o SMOTE). We find that *eWarn* performs better than its variant without SMOTE for all the studied online service systems. Without SMOTE, the F1-score of *eWarn* drops about 0.04 ~ 0.16, demonstrating the contribution of dealing with the data imbalance problem in *eWarn*.

To sum up, our classification model building component in *eWarn* is effective in both building a classifier via XGBoost and handling the data imbalance problem.

4.4 Efficiency

As a real-time incident prediction approach, time efficiency is a vital factor. For incoming alert data, if the prediction result cannot be provided in time, it could cause that engineers cannot take immediate actions to prevent the incident. Table 6 presents the average time costs of *eWarn* and three compared approaches for all the online service systems, including offline learning time and online prediction time. We find that the average online prediction time of all these approaches are quite small, i.e., no more than 1.13 seconds. That is, it is very efficient for *eWarn* to provide the prediction result for the current observation window. Although the training time of *eWarn* (i.e., 9.07 minutes) is slightly larger than AirAlert (i.e., 3.86 minutes) and FP-Growth (i.e., 2.70 minutes), the time cost is actually acceptable since the training process is conducted offline. Among these approaches, TF-IDF-LSTM has significantly longer training time (up to 32.30 minutes), which could take up many resources during training and cannot be conducive to efficiently update the learned model. Overall, *eWarn* has acceptable offline training time and negligible online prediction time, indicating that *eWarn* is indeed practical in the real world.

Table 6: Average time cost comparison between *eWarn* and compared approaches

Approaches	Offline learning (min)	Online prediction (s)
eWarn	9.07	0.04
AirAlert	3.86	0.06
TF-IDF-LSTM	32.30	1.13
FP-Growth	2.70	0.04

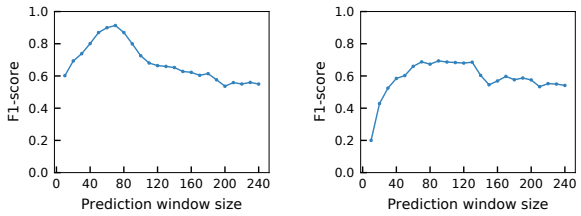

Figure 5: Two examples to show the effect of different prediction windows sizes (minutes) on F1-score

4.5 Effect of Parameters

Some parameters in *eWarn* have been introduced in § 4.1.3. Here, we mainly discuss the effect of the prediction window size on prediction performance. As mentioned in § 2.3, the prediction window size (t_p) is a vital parameter since it directly affects the labeling quality. Due to the space limit, we took two online service systems as examples to illustrate the influence of t_p on the performance of *eWarn*, whose results are shown in Figure 5. In the figure, the x-axis represents different prediction window sizes while the y-axis represents F1-score. Intuitively, the size of prediction windows has a significant influence on the performance of *eWarn*, since it directly affects the quality of labeling. More specifically, too large t_p may lead to an increase of false-positive observation windows during labeling, while too small t_p may lead to missing some positive windows. From Figure 5, the influence of t_p is indeed confirmed, and also the best t_p varies for the incidents of different systems. For example, the best t_p for S2 and S8 is 70 and 90, respectively. Therefore, it is necessary to set an appropriate prediction window size for each system. In our study, we identified the best prediction window size based on the performance on the validation set for each system, which is also applicable in practice since the historical data is usually accessible, especially in large companies. More discussions about the prediction window size can be found in § 5.2.

5 DISCUSSION

5.1 Success Stories

5.1.1 Incident Prediction. We have successfully applied *eWarn* to two top commercial banks (i.e., $\mathcal{A}$ and $\mathcal{B}$) in the country. Based on the feedback from engineers, *eWarn* is appreciated to be able to successfully assist them to anticipate incidents in advance, so that proactive actions can be taken to prevent system unavailability. In the following, we present four success cases collected from practice.

Case I: Long service response time caused by slow SQL. Response time is a key performance indicator of a service, and long response time would destroy user experience. Root cause of this

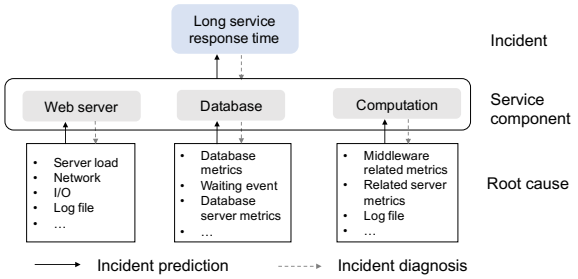

Figure 6: An example to illustrate the relationship between incident prediction and incident diagnosis

incident is database index fault, so it took a long time to execute several large SQL statements. Before the incident happens, *eWarn* successfully forecasted it based on database related alerts.

Case II: Frequent failed transactions caused by fullGC. Frequent failed transactions in banks would lead to user complaint and huge economic loss. More specifically, this incident was caused by buggy code imported by a recent software change, which created many objects occupying too much heap space and led to frequent fullGC. It was successfully predicted in advance by *eWarn* based on some resource metric alerts (e.g., memory usage and JVM heap usage) and JVM garbage collection (GC) log alerts.

Case III: Success rate of service degradation due to unexpected stress. For business in banks, low success rate is critical and would affect the amount of transactions, leading to economic loss. Due to an unexpected burst of user requests, CPU utilizations of related servers were influenced. *eWarn* effectively captured the omen patterns and predicted the incident accurately in advance.

Case IV: Server hang-up due to scheduled tasks. Server hang-up is a common incident in the real world because of system overload, which would cause related tasks on this server to fail. In this case, running scheduled tasks led to system overload, and the server was unresponsive. This incident was predicted accurately based on the alerts of related server resource metrics (e.g., I/O waiting time and CPU utilization).

In summary, *eWarn* has the ability to identify omen patterns from alert data and forecast incidents accurately in practice, which can earn some bonus time for engineers to take some proactive actions to prevent the incidents as soon as possible.

5.1.2 Incident Diagnosis. Although *eWarn* is designed for incident prediction, it can also assist engineers for incident diagnosis. Even though interpretable results provided by *eWarn* may not directly pinpoint the root cause (e.g. hardware errors, misconfigurations, or software bugs) of an incident, they can provide useful clues to narrow down the search space of diagnosis. When *eWarn* gives a positive result, the interpretable analysis component will tell engineers which feature makes the largest contribution to the predicted result. The most important feature may be closely related to the root cause of the incident.

Figure 6 takes case I above as an example to illustrate the relationship between incident prediction and incident diagnosis. *eWarn* identifies the omen patterns from alert data to predict the incident of "long service response time" precisely. Also, *eWarn* tells engineers the most important feature (i.e., topic#27 with keywords "Oracle", "AAS (average active session)", "SQL", etc., shown in Figure 3). In this way, engineers can infer that the possible root cause is related to database, which indeed narrows down the search space of diagnosis. Subsequently, they can take further actions to locate the root cause in database and prevent the incident as soon as possible.

5.2 Lessons Learned

We summarize some lessons learned from our study.

Not all incidents can be predicted well in advance. In spite of the effectiveness of *eWarn* in predicting incidents, we have to admit that not all incidents can be predicted well in advance. More specifically, some sudden incidents caused by unexpected factors (e.g., power outages) are difficult to be predicted. In our study, we find that application-level (high-level) incidents are much easier to be predicted. This is because high-level incidents are generally caused by faults in low-level components, e.g., server, database and network. Thus, *eWarn* can identify omen patterns from low-level alerts to predict high-level incidents.

Prediction window size is important for incident prediction. As presented in § 2.3 and § 4.5, the size of prediction windows (t_p) has a significant influence on the performance of *eWarn*, since it directly affects the quality of labeling. In general, increasing the value of t_p may increase the probability that an incident is predicted (e.g., when t_p is set to $+\infty$, simply predicting that an incident will occur would be always correct). If t_p is too large, the prediction is useless since it is not clear when exactly the incident will occur. Therefore, it is important for incident prediction to set an appropriate prediction window size.

Incremental Updating. Online service systems are complex and dynamic as developers continuously commit code and introduce new features. Consequently, new types of alerts and new omen patterns of incidents may be introduced accordingly. This may cause significant performance degradation if the prediction model is trained based on a fixed set of historical data. To make *eWarn* adapt to the dynamic environment and maintain the prediction performance, we build an incremental training pipeline where our model is trained with newly arriving alert data and incidents periodically, so that new patterns can be captured properly in time. Besides, retraining a model on new data from scratch suffers from high costs, but our XGBoost-based approach can be incrementally updated to achieve stably good performance with negligible cost.

5.3 Threats to Validity

We identify the following threats to validity in our study:

Subject systems: In our study, we used alert data and incidents from 11 different systems in a large commercial bank $\mathcal{A}$. However, the results might not represent other systems from other companies. Actually, it is very challenging to get access to data from companies. The bank we collaborate is very large-scale, which supports more than one hundred million users and includes hundreds of service systems, and thus we believe our results can demonstrate the value of our proposed approach. Moreover, we have applied *eWarn* to another large bank $\mathcal{B}$ (besides $\mathcal{A}$) presented in § 5.1, further demonstrating the generality of *eWarn*. In the future, we will reduce this threat by evaluating *eWarn* on more subject systems.

Measurements: To demonstrate the effectiveness of *eWarn*, we used precision/recall/F1-score as measurements, which have been

widely used in the prediction problem [42, 50]. Besides, we also consider the efficiency of *eWarn*. In the future, we will reduce this threat by considering more comprehensive metrics, such as precision-recall curve, FPR/TPR metrics, and how far ahead we can raise a true early warning.

Noisy in labeling: For a well-performing machine learning model, the training set and testing set should follow the same data distribution. However, some incidents are caused by some other unexpected or external factors, where there are no omen patterns before the incidents. These cases violate the assumption of independent and identical distribution and cannot be effectively predicted in advance. In our study, we ignored the effect caused by other factors and regarded them as positive samples. In the future, we will reduce this threat by incorporating detailed information of incident tickets and engineers' experience to remove noisy positive labels.

6 RELATED WORK

Incident Prediction. The most related work to ours is AirAlert [18], which also aims to predict general incidents based on alert data. However, it just simply considers the number of different kinds of alerts and does not deal with noisy alerts, which has been demonstrated to be *ineffective* in our study. Different from it, *eWarn* systematically overcomes main challenges in alert-based incident prediction by careful feature engineering, MIL for bypassing noisy alerts, and an interpretable generated report for a prediction result, significantly outperforming AirAlert in our study.

There are also a great deal of efforts spent on *predicting a specific type of failures* (which tend to be designed for a specific domain). Moreover, these works utilize system logs or metrics for prediction. For example, Prefix proposed in [50] utilizes switch log data to predict switch failures, CDEF [48] and MING [30] utilizes related metrics to predict disk failures and node failures, respectively. It is common for a large-scale system to generate tens of TBs of logs and more than thousands of metrics per day [31], leading to bringing great challenges to learn and incrementally update a prediction model and consuming a large amount of storage space and computing resources. Different from them, our work proposes a novel approach to *predicting general incidents* based on lightweight alert data, which are generated to report anomalies from other monitoring data such as metrics and logs and thus avoid processing massive raw monitoring data. Our experimental results have demonstrated the effectiveness of *eWarn*.

Alert Management. In recent years, tremendous efforts have been devoted into alert management in both academia and industry. As introduced in § 2.1, alert management techniques can be categorized into three stages: alert pre-processing, alert diagnosis and alert post-processing. Existing works mainly focus on how to reduce the number of alerts (alert pre-processing). This is because various service components could generate an overwhelming number of alerts and the majority of alerts are duplicate or correlated. The popular techniques of alert reduction include alert correlation [35] and alert clustering [29, 47, 52]. There are also some works about alert prioritization [24, 53], which aims to recommend severe alerts to engineers, because the number of alerts is much more than what engineers can properly investigate in practice. Different from them, our work aims to utilize alert data to conduct incident prediction.

7 CONCLUSION

We propose *eWarn*, a novel approach to predicting general incidents based on alert data, so as to take proactive actions to prevent the incoming incidents and ensure the quality of software services. Three key ideas of *eWarn* are an effective feature engineering component to deal with complex alert data, multi-instance learning to handle noisy alerts, and interpretable analysis to generate an interpretable report about the prediction result to facilitate the understanding and handling of incidents. An extensive study on 11 online service systems in a large commercial bank demonstrates the effectiveness of *eWarn* (the average F1-score of 0.82), outperforming three compared approaches. Besides, our results also confirm the contributions of main components in *eWarn*. In particular, real deployments of *eWarn* in the real world further demonstrating its practical performance.

ACKNOWLEDGEMENT

We thank the anonymous reviewers for their valuable feedbacks. 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] Average cost per hour of enterprise server downtime worldwide in 2019. https://www.statista.com/statistics/753938/worldwide-enterprise-server-hourly-downtime-cost/. [Online; accessed 04-Mar-2020].
[2] Incident Management. https://en.wikipedia.org/wiki/Incident_management_(ITSM). [Online; accessed 04-Mar-2020].
[3] Keras. https://keras.io/. [Online; accessed 04-Mar-2020].
[4] NumPy. https://numpy.org/. [Online; accessed 04-Mar-2020].
[5] pandas. https://pandas.pydata.org/. [Online; accessed 04-Mar-2020].
[6] scikit-learn. https://scikit-learn.org/.
[7] XGBoost. https://xgboost.readthedocs.io/. [Online; accessed 04-Mar-2020].
[8] Charu C Aggarwal and ChengXiang Zhai. 2012. *Mining text data*. Springer Science & Business Media.
[9] Christopher M Bishop. 2006. *Pattern recognition and machine learning*. springer.
[10] David M Blei, Andrew Y Ng, and Michael I Jordan. 2003. Latent dirichlet allocation. *Journal of machine Learning research* 3, Jan (2003), 993–1022.
[11] Marc-André Carbonneau, Veronika Cheplygina, Eric Granger, and Ghyslain Gagnon. 2018. Multiple instance learning: A survey of problem characteristics and applications. *Pattern Recognition* 77 (2018), 329–353.
[12] Marc-André Carbonneau, Eric Granger, and Ghyslain Gagnon. 2018. Bag-level aggregation for multiple-instance active learning in instance classification problems. *IEEE transactions on neural networks and learning systems* 30, 5 (2018), 1441–1451.
[13] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. 2002. SMOTE: synthetic minority over-sampling technique. *Journal of artificial intelligence research* 16 (2002), 321–357.
[14] Junjie Chen, Xiaoting He, Qingwei Lin, Yong Xu, Hongyu Zhang, Dan Hao, Feng Gao, Zhangwei Xu, Yingnong Dang, and Dongmei Zhang. 2019. An empirical investigation of incident triage for online service systems. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 111–120.
[15] Junjie Chen, Xiaoting He, Qingwei Lin, Hongyu Zhang, Dan Hao, Feng Gao, Zhangwei Xu, Yingnong Dang, and Dongmei Zhang. 2019. Continuous incident triage for large-scale online service systems. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 364–375.
[16] Junjie Chen, Shu Zhang, Xiaoting He, Qingwei Lin, Hongyu Zhang, Dan Hao, Yu Kang, Feng Gao, Zhangwei Xu, Yingnong Dang, and Dongmei Zhang. 2020. How Incidental are the Incidents? Characterizing and Prioritizing Incidents for Large-Scale Online Service Systems. In *The 35th IEEE/ACM International Conference on Automated Software Engineering*. to appear.
[17] Tianqi Chen and Carlos Guestrin. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. ACM, 785–794.
[18] Yujun Chen, Xian Yang, Qingwei Lin, Hongyu Zhang, Feng Gao, Zhangwei Xu, Yingnong Dang, Dongmei Zhang, Hang Dong, Yong Xu, et al. 2019. Outage Prediction and Diagnosis for Cloud Service Systems. In *The World Wide Web*

Conference. ACM, 2659–2665.

[19] Rui Ding, Hucheng Zhou, Jian-Guang Lou, Hongyu Zhang, Qingwei Lin, Qiang Fu, Dongmei Zhang, and Tao Xie. 2015. Log2: A cost-aware logging mechanism for performance diagnosis. In *2015 {USENIX} Annual Technical Conference ({USENIX}{ATC} 15)*. 139–150.

[20] John C Gower and Gavin JS Ross. 1969. Minimum spanning trees and single linkage cluster analysis. *Journal of the Royal Statistical Society: Series C (Applied Statistics)* 18, 1 (1969), 54–64.

[21] Jiawei Han, Jian Pei, and Micheline Kamber. 2011. *Data mining: concepts and techniques*. Elsevier.

[22] Jiawei Han, Jian Pei, and Yiwen Yin. 2000. Mining frequent patterns without candidate generation. *ACM sigmod record* 29, 2 (2000), 1–12.

[23] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[24] Guofei Jiang, Haifeng Chen, Kenji Yoshihira, and Akhilesh Saxena. 2011. Ranking the importance of alerts for problem determination in large computer systems. *Cluster Computing* 14, 3 (2011), 213–227.

[25] Jiajun Jiang, Weihai Lu, Junjie Chen, Qingwei Lin, Pu Zhao, Yu Kang, Hongyu Zhang, Yingfei Xiong, Feng Gao, Zhangwei Xu, Yingnong Dang, and Dongmei Zhang. 2020. How to Mitigate the Incident? An Effective Troubleshooting Guide Recommendation Technique for Online Service Systems. In *The 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Industry track*. to appear.

[26] Armand Joulin, Edouard Grave, Piotr Bojanowski, Matthijs Douze, Hérve Jégou, and Tomas Mikolov. 2016. FastText.zip: Compressing text classification models. *arXiv preprint arXiv:1612.03651* (2016).

[27] Yoon Kim. 2014. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882* (2014).

[28] Zhijing Li, Zihui Ge, Ajay Mahimkar, Jia Wang, Ben Y Zhao, Haitao Zheng, Joanne Emmons, and Laura Ogden. 2018. Predictive Analysis in Network Function Virtualization. In *Proceedings of the Internet Measurement Conference 2018*. 161–167.

[29] Derek Lin, Rashmi Raghu, Vivek Ramamurthy, Jin Yu, Regunathan Radhakrishnan, and Joseph Fernandez. 2014. 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*. ACM, 1630–1639.

[30] Qingwei Lin, Ken Hsieh, Yingnong Dang, Hongyu Zhang, Kaixin Sui, Yong Xu, Jian-Guang Lou, Chenggang Li, Youjiang Wu, Randolph Yao, et al. 2018. 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*. ACM, 480–490.

[31] Jinyang Liu, Jieming Zhu, Shilin He, Pinjia He, Zibin Zheng, and Michael R Lyu. 2019. Logzip: Extracting Hidden Structures via Iterative Clustering for Log Compression. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 863–873.

[32] Jian-Guang Lou, Qingwei Lin, Rui Ding, Qiang Fu, Dongmei Zhang, and Tao Xie. 2013. Software analytics for incident management of online services: An experience report. In *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 475–485.

[33] Jian-Guang Lou, Qingwei Lin, Rui Ding, Qiang Fu, Dongmei Zhang, and Tao Xie. 2017. Experience report on applying software analytics in incident management of online service. *Automated Software Engineering* 24, 4 (2017), 905–941.

[34] 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.

[35] Seyed Ali Mirheidari, Sajjad Arshad, and Rasool Jalili. 2013. Alert correlation algorithms: A survey and taxonomy. In *Cyberspace Safety and Security*. Springer, 183–197.

[36] Nikolaos Pappas and Andrei Popescu-Belis. 2014. Explaining the stars: Weighted multiple-instance learning for aspect-based sentiment analysis. In *Proceedings of the 2014 Conference on Empirical Methods In Natural Language Processing (EMNLP)*. 455–466.

[37] Parivash Pirasteh, Slawomir Nowaczyk, Sepideh Pashami, Magnus Löwenadler, Klas Thunberg, Henrik Ydreskog, and Peter Berck. 2019. Interactive feature extraction for diagnostic trouble codes in predictive maintenance: A case study from automotive domain. In *Proceedings of the Workshop on Interactive Data Mining*. 1–10.

[38] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. "Why should i trust you?" Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 1135–1144.

[39] Michael Röder, Andreas Both, and Alexander Hinneburg. 2015. Exploring the space of topic coherence measures. In *Proceedings of the eighth ACM international conference on Web search and data mining*. ACM, 399–408.

[40] Felix Salfner, Maren Lenk, and Miroslaw Malek. 2010. A survey of online failure prediction methods. *ACM Computing Surveys (CSUR)* 42, 3 (2010), 1–42.

[41] Mohammed Shatnawi and Mohamed Hefeeda. 2015. Real-time failure prediction in online services. In *2015 IEEE Conference on Computer Communications (INFOCOM)*. IEEE, 1391–1399.

[42] Ruben Sipos, Dmitriy Fradkin, Fabian Moerchen, and Zhuang Wang. 2014. Log-based predictive maintenance. In *Proceedings of the 20th ACM SIGKDD international conference on knowledge discovery and data mining*. 1867–1876.

[43] J Wang, C Li, S Han, S Sarkar, and X Zhou. 2017. Predictive maintenance based on event-log analysis: A case study. *IBM Journal of Research and Development* 61, 1 (2017), 11–121.

[44] Lingzhi Wang, Nengwen Zhao, Junjie Chen, Pinnong Li, Wenchi Zhang, and Kaixin Sui. 2020. Root-Cause Metric Location for Microservice Systems via Log Anomaly Detection. In *The 2020 IEEE International Conference on Web Services*. to appear.

[45] Gerhard Widmer and Miroslav Kubat. 1996. Learning in the presence of concept drift and hidden contexts. *Machine learning* 23, 1 (1996), 69–101.

[46] Haowen Xu, Wenxiao Chen, Nengwen Zhao, Zeyan Li, Jiahao Bu, Zhihan Li, and et.al. 2018. Unsupervised Anomaly Detection via Variational Auto-Encoder for Seasonal KPIs in Web Applications. In *WWW*.

[47] Jingmin Xu, Yuan Wang, Pengfei Chen, and Ping Wang. 2017. Lightweight and Adaptive Service API Performance Monitoring in Highly Dynamic Cloud Environment. In *2017 IEEE International Conference on Services Computing (SCC)*. IEEE, 35–43.

[48] Yong Xu, Kaixin Sui, Randolph Yao, Hongyu Zhang, Qingwei Lin, Yingnong Dang, Peng Li, Keceng Jiang, Wenchi Zhang, Jian-Guang Lou, et al. 2018. Improving service availability of cloud systems by predicting disk error. In *2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18)*. 481–494.

[49] Ke Zhang, Jianwu Xu, Martin Renqiang Min, Guofei Jiang, Konstantinos Pelechrinis, and Hui Zhang. 2016. Automated IT system failure prediction: A deep learning approach. In *2016 IEEE International Conference on Big Data (Big Data)*. IEEE, 1291–1300.

[50] Shenglin Zhang, Ying Liu, Weibin Meng, Zhiling Luo, Jiahao Bu, Sen Yang, Peixian Liang, Dan Pei, Jun Xu, Yuzhi Zhang, et al. 2018. Prefix: Switch failure prediction in datacenter networks. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 2, 1 (2018), 2.

[51] Xu Zhang, Yong Xu, Qingwei Lin, Bo Qiao, Hongyu Zhang, Yingnong Dang, Chunyu Xie, Xinsheng Yang, Qian Cheng, Ze Li, et al. 2019. Robust log-based anomaly detection on unstable log data. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 807–817.

[52] Nengwen Zhao, Junjie Chen, Xiao Peng, Honglin Wang, Xinya Wu, Yuanzong Zhang, and et al. 2020. Understanding and Handling Alert Storm for Online Service Systems. In *2020 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. ACM.

[53] Nengwen Zhao, Panshi Jin, Lixin Wang, Xiaoqin Yang, Rong Liu, Wenchi Zhang, Kaixin Sui, and Dan Pei. 2020. Automatically and Adaptively Identifying Severe Alerts for Online Service Systems. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2420–2429.

[54] Nengwen Zhao, Jing Zhu, Yao Wang, Minghua Ma, Wenchi Zhang, Dapeng Liu, Ming Zhang, and Dan Pei. 2019. Automatic and Generic Periodicity Adaptation for KPI Anomaly Detection. *IEEE Transactions on Network and Service Management* 16, 3 (2019), 1170–1183.

[55] Xiang Zhou, Xin Peng, Tao Xie, Jun Sun, Chao Ji, and et al. 2019. Latent error prediction and fault localization for microservice applications by learning from system trace logs. In *ESEC/FSE*. ACM, 683–694.

[56] Xiang Zhou, Xin Peng, Tao Xie, Jun Sun, Chao Ji, Wenhai Li, and Dan Ding. 2018. Fault analysis and debugging of microservice systems: Industrial survey, benchmark system, and empirical study. *IEEE Transactions on Software Engineering* (2018).