




Smart Issue Detection for Large-Scale Online Service Systems Using Multi-Channel Data

Liushan Chen¹(✉), Yu Pei², Mingyang Wan¹, Zhihui Fei¹, Tao Liang¹,
and Guojun Ma¹

¹ ByteDance Inc., Shenzhen, China
chenliushan@bytedance.com

² Department of Computing, The Hong Kong Polytechnic University, Hong Kong,
Hong Kong S.A.R., China

Abstract. Given the scale and complexity of large online service systems and the diversity of environments in which the services are to be invoked, it is inevitable that those service systems contain bugs that affect the users. As a result, it is essential for service providers to discover issues in their systems based on information gathered from users. iFeedback is a state-of-the-art technique for user-feedback-based issue detection. While it has been deployed to help detect issues in real-world service systems, the accuracy of iFeedback's detection results is relatively low due to limitations in its design. In this paper, we propose the SKYNET technique and tool that analyzes both user feedback gathered via specific channels and public posts collected from social media platforms to more accurately detect issues in service systems. We have applied the tool to detect issues for three real-world, large-scale online service systems based on their historical data gathered over a ten-month period of time. SKYNET reported in total 2790 issues, among which 93.0% were confirmed by developers as reflecting real problems that deserve their close attention. It also detected 58 out of the 62 severe issues reported during the period, achieving a recall of 93.5% for severe issues. Such results suggest SKYNET is both effective and accurate in issue detection.

1 Introduction

Large-scale online service systems are becoming indispensable for people's work and everyday life nowadays. They also get more and more complex so as to support the ever-growing needs of their users for new and more powerful functionalities. The scale and complexity of such services as well as the diversity of environments in which the services are to be invoked, however, have made it more challenging than ever for developers to make sure the services will always behave as expected. Despite the tremendous amount of time and effort developers invest in testing and debugging such online service systems, it is almost inevitable that some bugs escape the developers' attention, get released into the field, and negatively impact users' experience with the services. It is, therefore, extremely important for the service providers to discover issues in their systems based on information gathered from users in a timely manner.

© The Author(s) 2024

D. Beyer and A. Cavalcanti (Eds.): FASE 2024, LNCS 14573, pp. 165–187, 2024.

https://doi.org/10.1007/978-3-031-57259-3_8

In view of that, Zheng et al. [45] recently proposed the iFeedback approach to detecting issues based on user feedback. While the approach has been deployed to help detect issues in large-scale online service systems and has successfully detected severe issues, the overall precision of its results is relatively low, 76.2% to be exact [45]. We conjecture there are three reasons for that. First, iFeedback extracts word combinations from feedback texts as indicators of issues. Since word combinations only capture the lexical, rather than semantical, characteristics of feedback texts, they, as issue indicators, tend to be overly sensitive to the wording of user feedback. Second, iFeedback detects anomalies at the level of time intervals based on all the user feedback gathered during those intervals, which is too coarse-grained. Since a wide range of different types of user feedback, concerning issues or not, may get reported during each time interval, it is more likely for iFeedback’s judgment to be influenced or even misled by user feedback that does not report any issues. Third, iFeedback applies an unsupervised algorithm to cluster the feedback during anomalous time intervals based on the word combinations and their contexts. While unsupervised clustering algorithms are less expensive to apply, they tend to produce less precise results than supervised algorithms in general [36].

To address these limitations of iFeedback and improve the quality of issue detection results, we propose in this paper a novel approach, named SKYNET, to automatically detecting issues in online service systems based on multi-channel user input, including both user feedback and messages posted on social media platforms. More concretely, SKYNET first employs a cascading classifier to label the user feedback texts based on an input hierarchical label system for different types of user experiences. Then, it applies time-series data analysis to predict, based on historical data, a threshold for the normal frequencies of user feedback reporting each known type of negative user experience; and it reports an issue when more feedback of the same type than allowed by the threshold is gathered from the users. Meanwhile, for user feedback reporting negative experiences of previously unknown types, SKYNET reports an issue when an abnormous amount of such user feedback concerns similar negative user experiences. The semantic embedding of feedback texts and the customized issue detection process adopted by SKYNET enables it to detect more real issues in service systems and to prune out most false positives. In view that social media platforms have become important and popular venues for users to share their experiences with various services and products, SKYNET also monitors and analyzes messages posted on social media platforms to detect issues before they generate a large number of user feedback or attract considerable unwanted public attention.

We have implemented the SKYNET approach into a tool with the same name. To empirically evaluate SKYNET’s effectiveness, we applied it to detect issues for three real-world, large-scale online service systems based on their historical data gathered from a ten-month duration. SKYNET reported in total 2790 issues, 93.0% of which were confirmed by operators and developers as reflecting real problems that deserve their close attention. Besides, SKYNET was able to detect

58 of the 62 severe issues that occurred during that period of time. Such results suggest SKYNET is highly effective and accurate in issue detection.

Contributions. This paper makes the following contributions:

- We propose the SKYNET technique that analyzes both user feedback gathered from specific channels and public posts collected from social media platforms to accurately detect issues in large-scale online service systems.
- We develop SKYNET into a tool with the same name.
- We empirically evaluate SKYNET by applying it to detect issues for three real-world service systems based on historical data. The results produced suggest that SKYNET is highly effective and accurate.

2 Related Work

Our work is closely related to existing work in the following areas.

Anomaly detection based on backend monitoring. In view that many issues in online service systems affect performance attributes like “disk queue length” and “network retransmission rate” of the backend systems, people often monitor the corresponding key performance indicators (KPIs) of the systems and rely on the values to detect anomalies in those services [15,18,21,22,23,25,26,39,44]. For instance, Laptev et al. [21] proposed the EGADS system that combines a collection of anomaly detection and forecasting models to detect anomalies in time-series KPI data. Liu et al. [25] proposed the Opprentice system that trains a random forest with labeled KPI features to select appropriate parameters and thresholds for existing detectors. Xu et al. [44] proposed an unsupervised anomaly detection algorithm, named Donut, to effectively detect anomalies in seasonal KPIs. Given that online service systems automatically generate issue reports and alerts when the monitored indicators exhibit anomalous values, techniques have also been developed to mine attribute collections of issue reports [15,24] to characterize and detect incidents [22].

Issue detection based on user feedback. Many issues, e.g., user interface defects and silent back-end issues, in those systems, however, are not reflected by pre-defined KPIs [45]. In view of that and the fact that user opinions coming in different forms (e.g., user feedback, tweets, and forum posts) contain valuable information to support software development and maintenance [12,13,29,30,41,42], Zheng et al. [45] proposed the iFeedback approach to detecting issues based on user feedback on-the-fly. iFeedback first extracts word combination-based indicators to represent an issue and collects each indicator’s historical occurrence trend (HOT), then the long-term and short-term windows of the HOTs are fed to a binary classifier to identify anomalous time intervals, and in the end, user feedback from time intervals containing issues are clustered as reporting different issues. SKYNET improves on iFeedback from three perspectives. First, iFeedback extracts word combinations from feedback texts as indicators of issues, which captures only the lexical characteristics of feedback texts, while SKYNET employs the ALBERT-tiny model to encode user feedback so that the semantics of user feedback can be taken into account during the issue detection process.

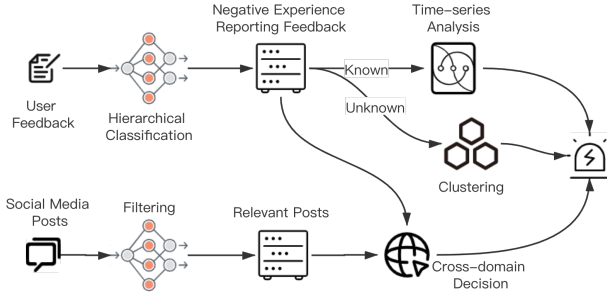


Fig. 1: An overview of the issue detection process with SKYNET.

Second, iFeedback detects anomalies at the level of time intervals based on all the gathered user feedback, which is often too coarse-grained and increases the chance of coincident non-issue-reporting feedback influencing and misleading the issue detection process. In contrast, SKYNET employs a cascading classification algorithm to label user feedback based on a hierarchical label system and only takes feedback that reports negative user experiences into account in the remaining issue detection process. Third, SKYNET also monitors and analyzes messages posted on social media platforms to detect issues in a timely manner, which complements user-feedback-based issue detection.

Learning from user opinions in other forms. User opinions in other forms have also been utilized to support various types of activities in software development. Gao et al. [14] proposed the IDEA framework that detects issues from review texts of apps. Stanik et al. [38] proposed an approach to identify aspects of software systems to improve based on user comments received on Twitter. While those identified aspects may indeed need improvement, they not necessarily are issues in the corresponding software systems. Guzman et al. [16] proposed the ALERTme approach that automatically classifies, groups, and ranks tweets to facilitate the analysis of application-related tweets. Williams and Mahmoud [43] conducted a study on leveraging Twitter as a main source of software user requirements. Johann et al. [19] proposed the SAFE approach that extracts keywords from app feature descriptions written by developers and app reviews on app stores to better characterize the apps. Compared with these works, SKYNET focuses on detecting issues in online service systems based on user feedback and social media posts.

3 The SkyNet Approach

Figure 1 depicts an overview of the issue detection process with SKYNET. SKYNET leverages deep learning algorithms to detect issues based on multi-channel data and it combines two loosely coupled processes: The main process is designed for detecting issues based on user feedback texts gathered through dedicated channels that are embedded in the service systems, while the auxiliary process

complements the main process and aims to detect issues using posts collected from social media platforms. Each issue detected by SkyNet is associated with a collection of user feedback, a social media post in case it is the main concern of the post, and a list of ten keywords extracted from the user feedback and post using the TF-IDF method [6]. While the keywords help provide a rough idea about an issue, developers must examine the associated user input to determine whether the reported issues reflect real problems in the service systems. In the rest of this section, we explain in detail the steps in SKYNET’s main and auxiliary issue detection processes.

Note that, as in other model-based approaches, we periodically review the input user feedback and social media posts as well as the detected issues, manually rectify the incorrect detection results if any, and use the new data to fine-tune the models that SKYNET utilizes so as to keep the models fit for the updated business situation and to prevent model degradation. Also note that, although sometimes users include images in their feedback and social media posts to help explain the problems they have encountered, SKYNET does not utilize such information in its current implementation. We leave the development of new techniques that exploit the extra image information to facilitate issue detection for future work.

3.1 Hierarchical Classification of User Feedback

The first step in issue detection with SKYNET is to decide the type of user experience that each piece of the gathered user feedback reports. SKYNET makes such decisions on the basis of a hierarchical label system, where the labels characterize with different levels of detail the types of (negative) user experiences that users report in their feedback.

SKYNET differentiates three broad categories of user feedback in issue detection, namely feedback reporting negative user experiences of a known type, feedback reporting negative user experiences of unknown types, and feedback not reporting negative user experiences. User feedback from the first two categories is collectively called *negative experience reporting feedback*. Note that not all negative user experiences are caused by issues in service systems. For example, although a user’s access to an online service will be blocked if her device is offline due to a hardware failure, the experience does not indicate anything problematic in the online service system.

Feedback Encoding Since SKYNET is designed to detect issues in large-scale online service systems, and it may need to process a large number of user feedback under tight time constraints, we use ALBERT-Tiny [20] to encode the user feedback. BERT [11] is a pre-trained state-of-the-art language representation neural network model with strong semantic comprehension capability. ALBERT [20] is a lite BERT architecture, and it lowers the memory consumption and increases the training speed of BERT, while without significantly sacrificing BERT’s semantic comprehension ability, by sharing parameters across layers and reducing embedding dimensions of words. ALBERT-Tiny [20] is the smallest version of ALBERT that is 10x times faster than BERT for inference.

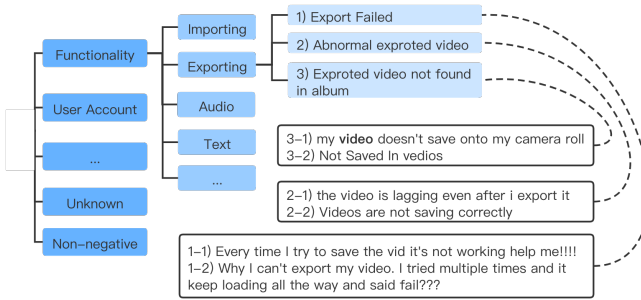


Fig. 2: A sample hierarchical label system (in blue) and some examples of the associated user feedback.

Hierarchical Label System To correctly decide which type of user experience each user feedback reports is crucial since incorrect decisions made here may mislead the downstream steps and cause the whole task of issue detection to fail. SKYNET employs an existing hierarchical label system to facilitate making those decisions. In the system, each label corresponds to a particular type of user experience that users may have with the target online service system.

Designing a label system to properly characterize user experiences is a challenging task. SKYNET adopts a hierarchical, rather than flat, label system mainly because it is extremely difficult, if not impractical, to decide *a priori* on the right granularity level for the labels in a flat system so as to strike a good balance between the accuracy and the value of the classification results based on that label system. On the one hand, a coarse-grained label system often makes it easier for a classifier to correctly label the input data, but the classification results may not be very useful since each label encodes little extra information. On the other hand, a fine-grained label system typically makes it harder for a classifier to correctly label the input data, but a correct label in this case can be highly valuable since it encodes abundant extra information. In the context of user feedback classification for issue detection, coarse-grained labels provide relatively vague information about the user experience, which may not be sufficient to help developers effectively confirm or understand the underlying issues.

Figure 2 displays part of the hierarchical label system that SKYNET uses for classifying the user feedback on an online video editing system. In the hierarchical label system, labels at the top level classify all the user feedback into broad categories concerning aspects like “Functionality” and “User Account” of the online system, labels at the intermediate level partition the broad categories into smaller, finer-grained ones, while labels at the bottom level correspond to specific types of experiences that users may have when using the online system. Two top-level labels in the hierarchical label system, namely “Unknown” and “Non-negative”, are special in the sense that they do not have subordinate labels because they are for user feedback texts that report negative user experiences of previously unknown types and that do not report negative user experiences,

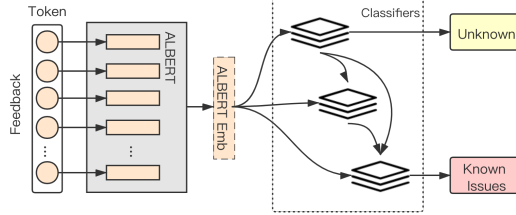


Fig. 3: The process of hierarchical user feedback classification in SKYNET.

respectively. Since some user experiences of previously unknown types may still reveal important issues of the systems, SKYNET conducts extra analysis on the related feedback to determine if they report any issues. Section 3.2 gives more details about the analysis. User feedback classified as “Non-negative” will not be further processed by SKYNET.

Figure 2 also lists some example feedback snippets from users of the online video editing system and associates the snippets to their corresponding labels. Two things from the examples are worth noting. First, users often use different words in describing the same issue. For example, the words “save” and “export” were used in snippets 1-1 and 1-2 to refer to the action of exporting a video, respectively. Second, different words with similar meanings may be used to describe user experiences of distinct types. For example, the word “save” was used in both snippets 2-2 and 3-2, which report different types of negative user experiences. Due to such flexibility in natural language expressions, using word combinations like (“save” and “video”) to characterize and group user feedback, as was done in previous work [45], may often produce results of low precision. In view of that, SKYNET extracts the semantics of the experiences reported in user feedback via deep learning and classifies user feedback based on their semantics.

We do not consider the requirement for an input hierarchy of user feedback labels as a major restriction to SKYNET’s applicability for two reasons. First, although not every service system readily has a dedicated hierarchy of user feedback labels, hierarchies from similar systems could be used instead to bootstrap the application of SKYNET on a new service system since, according to our experience, systems with similar functionalities often share hierarchies of user feedback labels. Second, a collection of appropriate issue labels is essential for the effective management of issues in large online service systems. Developers need to devise the labels with or without tool support, and the labels can be organized into a hierarchy to drive SKYNET. While the construction of such a hierarchical label system may require some manual effort, such investment is worthwhile in the long term since a high-quality label system can greatly improve the result accuracy of feedback classification and issue detection.

Cascading Classification SKYNET employs cascading classification to associate user feedback to the labels from the hierarchical label system. Cascading

is a particular case of ensemble learning based on the concatenation of several sub-classifiers [2]. In SKYNET’s cascading classification for hierarchical labels, each sub-classifier targets only the labels at a particular level, and the output of a high-level sub-classifier is used as additional input to drive lower-level sub-classifiers in the cascade. In such a setting, it is relatively easier for high-level sub-classifiers to produce proper classification results since the number of labels they need to consider is small and the differences between instances from different classes are big; It is also relatively easier for low-level sub-classifiers to achieve more precise classification results since they only need to focus on the labels subordinate to those labels output by high-level sub-classifiers [35].

Figure 3 shows the cascade classifier SKYNET employs to categorize the user feedback on the online video editing system described in Section 3.1. The classifier contains three sub-classifiers, each for one level of the label hierarchy. Each sub-classifier is a two-layer network, with the neural cells on each layer being fully connected with each other, and it takes all its parent-level classifiers’ output, if any, as input for the current level’s classification. For instance, the top-level sub-classifier classifies user feedback based on the highest level labels like “Functionality” and “User Account” according to the input text embedding. While the bottom-level sub-classifier takes both the text embedding and the output of the two sub-classifiers at higher levels as input to conduct the most fine-grained classification. The connections between classifiers help preserve the cascade relationship between multi-level labels and improve classification accuracy.

Particularly, each sub-classifier is a multi-class classifier with a loss function defined as $L = \frac{1}{N} \sum_{i=1}^N \sum_{c=1}^C \text{loss}(y_{ic}, \hat{y}_{ic})$, where N is the number of samples, C is the total number of classes in the classification, \hat{y}_{ic} is the probability of i th training example belonging to the c th class, y_{ic} is a binary indicator function that represents the ground truth label, while $\text{loss}(y_{ic}, \hat{y}_{ic})$ is the cross-entropy loss between the classification results and the ground truth. Cross-entropy loss [10] is a common loss function for classification tasks, and its value increases as the predicted probability diverges from the actual labels.

The loss function for the overall cascading classification model is defined as $L_{\text{overall}} = \alpha L_1 + \beta L_2 + \gamma L_3$. That is, the overall loss L_{overall} of the model is the weighted sum of the loss L_n at the n -th cascading level ($1 \leq n \leq 3$), with α, β and γ being the weights of corresponding levels. We assign decreasing values 0.8, 0.6, and 0.4, to α, β and γ , respectively, based on the intuition that an incorrect label at any level will lead to incorrect labels for all the underneath levels. With the cascading connections, the weight of the first level sub-classifier will be adjusted with respect to the loss of all classifiers at the three levels during back-propagation, and the weight of the second level sub-classifier will be adjusted with respect to the loss of sub-classifiers at the second and third levels.

3.2 Issue Detection Based on User Feedback

While it is useful to classify feedback texts based on the types of user experiences they report, it is neither necessary nor practical to manually examine all the user feedback that reports negative experiences. On the one hand, not all

user feedback reporting negative experiences is caused by issues in online service systems that demand manual inspection by developers. On the other hand, user feedback reporting negative experiences with popular service systems often comes in overwhelming numbers, and therefore it can be prohibitively expensive to manually handle all those user feedback.

To help developers better distribute their time and effort on tasks for issue handling, SKYNET only reports issues for negative experiences shared by a large number of users. Particularly, SKYNET employs a time series forecasting technique to dynamically predict a threshold for the frequency of each known type of negative user experience. An alert indicating the discovery of an issue that needs to be handled will be raised if negative user experiences of the related type get reported more often than allowed by the threshold.

Issues of Known Types When SKYNET classifies a piece of user feedback text to a known type of negative user experience, we say the feedback is an *instance* of the user experience type. By concatenating the instance numbers of a known negative user experience type within each time unit, we form time-series data about the frequency of that type of user experience. Based on the hypothesis that a rising issue of known type will cause outliers in the time-series data of its corresponding label, SKYNET determines that there is an issue when the number of user feedback reporting a particularly known type of negative experience in a time period exceeds a threshold.

Since the normal frequency of each type of negative user experience is closely related to several factors that vary across experience types and over time, adopting a fixed threshold for all negative user experience types would be too rigid. First, different types of negative experiences naturally occur in different frequencies. For example, in our experience, it is normal to have in each day a few hundred users of a large-scale service system reporting that they cannot receive the verification code, and the reasons often include things like typos in their phone numbers, unstable connections of their phones, and the low response speed of their network operators, none of which is indicative of issues in our systems. On the contrary, the daily number of users reporting problems with uploading files is typically much smaller, and when that number increases significantly, it is highly likely that an issue in our system is the cause. Second, the normal frequency of any type of negative user experience fluctuates at different times in a day, a week, or a month. For instance, most negative experiences occur more often during the day when most users are active than at midnight when most users have fallen asleep. Since predicting a dynamic threshold with historical data is a widely accepted way to detect issues [33,21], SKYNET naturally formulates the issue detection problem as a time series forecasting problem that predicts the normal frequency range for each label based on historical data.

More concretely, we apply a sliding window strategy for the segmentation of each label's historical data, and we adopt a classical bidirectional long short-term memory (BiLSTM) [17] network to learn the historical trends of individual labels. The window size is set to 50 time units in the current implementation, and

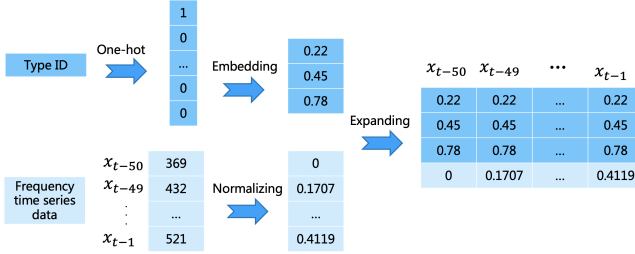


Fig. 4: Expansion of frequency data with feedback type ID, which enables the prediction of multiple thresholds with a unified BiLSTM model.

the window slides with a stride length of one time unit. Note that all outliers—data points outside the interquartile range [4]—in the time series are removed, the Min-Max normalization [31,32] is applied for feature scaling before training.

BiLSTM is a recurrent neural network that takes historical time series data as input to make a prediction based on the trend. To predict a value y'_t for time t , the model takes a series of historical data $[x_{t-50}, \dots, x_{t-1}]$ as input, where x_t represents the feature vector for the time unit immediately after t . During training, the model loss is the mean squared error between the actual value y_t and the predicted value y'_t for time t .

Based on the predicted frequency y'_t for a label, SKYNET calculates the threshold th_t for the label as $y'_t * dr$, where dr is a dynamic ratio calculated as $\log(\text{std}([x_{t-50}, \dots, x_{t-1}]) / \text{mean}([x_{t-50}, \dots, x_{t-1}]))$. The rationale behind the calculation of the threshold is that the magnitude of acceptable frequency fluctuations should be proportional to the absolute value of the frequency prediction for the label. For example, when the occurrence of a label increases by ten, this fluctuation would be relatively smaller if the label’s regular frequency y_t is ten thousand instead of a hundred. We apply a log transformation when calculating dr to keep it relatively small.

Predicting Multiple Thresholds with A Unified BiLSTM Model Usually, predicting the normal frequency of a particular type of user feedback requires training a specialized model with the historical frequency data associated with that type. Training one specialized model for each prediction task, however, would cause high costs for the application and maintenance of SKYNET. To reduce those costs, we expand the values in the time series data for each type of user feedback with the identity of that type and use the expanded time series data of all feedback types to train a unified BiLSTM model. The unified model is then able to predict the normal frequencies of different types of user feedback.

Particularly, we expand the feedback frequency data in three steps, as depicted in Figure 4. We first apply one-hot encoding to produce a unique value as the identity of each type of user feedback. Since one-hot type IDs generated in this way are typically sparse, we then transfer them to a dense vector via a fully-connected network $g(\cdot)$. Afterward, the frequency data and the dense vector will

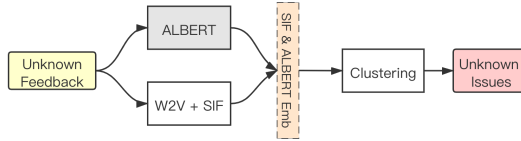


Fig. 5: Detecting issues of unknown types by clustering user feedback.

be combined to form the expanded frequency data. That is, given the one-hot ID δ of a user feedback type and the vectorized frequency \bar{x}_t of this user feedback type at time t , the expanded frequency is constructed as $\bar{x}_t \oplus g(\delta)$, where \oplus indicates vector concatenation. Here, the transfer of one-hot type IDs to dense vectors is necessary because, without it, all but one dimensions of the input data would be for the feedback type ID, and it will be extremely hard for the BiLSTM model to learn meaningful knowledge about the feedback frequency.

Evaluation results of SKYNET on three real-world large-scale online service systems, as detailed in Section 4, show that such unification does help improve the efficiency, while without significantly sacrificing the effectiveness, of threshold prediction in SKYNET.

Issues of Unknown Types Recall that all feedback reporting previously unknown types of negative user experiences will be classified into the “Unknown” category, and such feedback may also reveal issues if many of them concern similar experiences. In view of that, SKYNET clusters user feedback in category “Unknown” periodically (e.g., every half an hour) and raises an issue when the number of feedback in a cluster exceeds a threshold. Figure 5 depicts the main steps SKYNET takes to detect issues of unknown types based on clustering.

To increase the chance that user feedback reporting similar user experiences gets placed into one cluster, it is important that the embedding properly captures the semantic characteristics of the feedback texts. To that end, SKYNET naturally uses the fine-tuned ALBERT-Tiny model to generate the deep semantic embedding of these feedback texts. Feedback clustering solely based on that embedding, however, may suffer from the overfitting problem and miss issues of unknown types because the ALBERT-Tiny model was fine-tuned w.r.t. the input hierarchical label system. Therefore, SKYNET also incorporates the shallow semantics extracted with Word2Vec [27,28] and Smooth Inverse Frequency (SIF) [9] to facilitate the clustering. Word2Vec is a pre-trained model that masters word associations from a large corpus of text, while SIF uses the vector calculated as the weighted average of all word vectors to embed a sentence. Given a piece of feedback text, SKYNET first applies Word2Vec to produce the embedding for each token in the text and then converts the token embeddings to a sentence embedding with SIF. Afterward, the overall embedding of the feedback combining its shallow and deep semantic information is formed by concatenating the embeddings produced by ALBERT-Tiny and SIF, respectively.

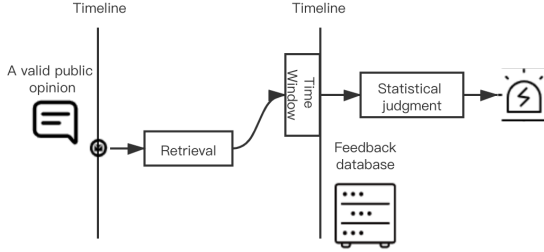


Fig. 6: Cross-domain decision mechanism. The valid public opinion is used to retrieve feedback according to both syntactic and semantic similarity from the database in a time window. The retrieved feedback results then go through a statistical judgment for issue alert.

With the overall semantic embedding as input, SKYNET employs the K-means algorithm to cluster “Unknown” feedback into groups. Note that, since the “Unknown” user feedback usually concerns a wide range of user experiences without concentrating on any specific types, we expect the resultant clusters to be small in size. Correspondingly, when those user feedback texts form large groups, it is highly likely that the feedback in those groups reveals issues in the system. Specifically, SKYNET reports an issue if the size of a cluster exceeds a threshold $H_f = \text{MAX}(N_{total}/m * \alpha, \beta)$, where N_{total} is the total number of feedback being clustered, m is the (predefined) number of clusters to produce, while both α and β are constants. In other words, an alert will be raised if the number of feedback in a cluster is larger than both α times the average cluster size and a fixed value β . We conservatively set α to 5 in SKYNET since, according to our experience, an issue often causes the size of its corresponding feedback cluster to increase by 10 times or even more. β is introduced to avoid reporting issues merely because the value of $N_{total}/m * \alpha$ is very small, e.g., when the total number of user feedback to be clustered is small, and we empirically set it to 10.

3.3 Issue Detection Based on Social Media Data

Due to the potentially high cost and the impact that negative public opinions may cause when they are overlooked, SKYNET dedicates an auxiliary process to detecting issues reflected by posts on social media platforms.

Compared with user feedback collected from dedicated channels that is more informative and has labeled historical data for training, social media posts usually contain noisy data, are less structured, and often cover a wide range of topics, making it more challenging to extract issue-related information from them. In view of that, SKYNET adopts a two-stage denoising process to prune out most posts that are either not directly related to the service system under consideration or not reporting experiences likely associated with issues.

More concretely, during the two-stage denoising process, SKYNET first applies keyword-based search to filter out posts that do not mention the name of

the target service system, and then applies a binary classification model constructed with ALBERT-Tiny to further filter out posts not reporting negative user experiences. To train the classification model, we collect product-related posts and manually labeled them to distinguish whether they report negative user experiences. We refer to all the social media posts that are retained after the two-stage denoising process as *relevant* posts.

To identify social media posts that report negative experiences likely associated with issues, SKYNET employs a cross-domain joint-decision-making process based on both user feedback and social media posts. As depicted in Figure 6, for each relevant social media post, SKYNET first retrieves similar user feedback from past time windows. We consider two types of similarities between user feedback and social media posts. The lexical similarity is calculated using the Lucene correlation algorithm that comes with ElasticSearch [3], which is based on the classic BM25 algorithm [8]. We consider a piece of user feedback to be a lexical match of a social media post if the BM25 score between them is higher than a threshold 40. The semantic similarity is calculated as the Euclidean distance between the ALBERT-Tiny embeddings of the user feedback and the social media post. We consider a piece of user feedback to be a semantic match of a social media post if the distance is smaller than a threshold of 0.4. A piece of user feedback is considered a match for a social media post if it is a lexical or semantic match for the post. Obviously, it is possible that a piece of user feedback is both a lexical and a semantic match of a social media post.

Given a relevant social media post p , let N_h and N_d be the total number of matching user feedback for p in the past hour and day, respectively, SKYNET raises an issue if N_h exceeds the threshold $H_h = \text{MAX}(\alpha_h * \overline{N}_h, \beta_h)$ or N_d exceeds the threshold $H_d = \text{MAX}(\alpha_d * \overline{N}_d, \beta_d)$, where \overline{N}_h and \overline{N}_d are the average number of matching user feedback for p in each hour and day of the past week, respectively, while α_h , α_d , β_h , and β_d are constants. Intuitively, an alert will be generated if (1) the number of similar user feedback in the past hour is larger than both α_h times the hourly average across the past week and a fixed value β_h or (2) the number of similar user feedback in the past day is larger than both α_d times the daily average across the past week and a fixed value β_d . We empirically assign 3, 3, 5, and 10 to α_h , α_d , β_h , and β_d , respectively, in the current implementation of SKYNET, and we leave the development of more sophisticated techniques for predicting the threshold values for future work.

4 Experimental Evaluations

We experimentally evaluated the effectiveness of SKYNET and the usefulness of its components based on its application results produced on real-world online service systems. Our evaluation aims to address the following research questions:

- RQ1: How effective is SKYNET in detecting issues in industry-level online service systems? In RQ1, we assess the effectiveness of SKYNET in issue detection in terms of the precision and recall it achieves from a user’s perspective.

Table 1: Industry-level online service systems used as the subjects in our experiments.

ID	DESCRIPTION	MAU	#FEEDBACK	#LABEL		
				TOP	INTERM.	BOTTOM
S1	An online video sharing platform	> 600m	> 100,000	36	140	360
S2	An online video editing system	> 130m	> 1,000	13	188	442
S3	An online beauty camera platform	> 27m	> 200	7	51	84

RQ2: How useful are the individual component mechanisms of SKYNET for the overall issue detection? Recall that SKYNET integrates three components to effectively detect issues in large-scale online service systems, namely a component C_k that applies cascading classification and time series analysis to detect issues of known types based on user feedback, a component C_u that applies the K-means clustering algorithm to detect issues of unknown types based on user feedback, and a component C_p that applies joint decision making to detect issues based on social media posts. In RQ2, we investigate how much each of these components contributes to the overall effectiveness of SKYNET.

We were not able to experimentally compare SKYNET with iFeedback for two reasons. First, the implementation of iFeedback is not publicly available. Second, faithfully re-building the tool is hardly viable because important information regarding its implementation is missing from the related publication. For example, we only know from the publication that iFeedback employs an XGBoost-based model to classify whether a time interval contains an issue, and it applies a hierarchical algorithm to cluster the user feedback as reporting different issues [45], but no information about the settings and parameters of the model and algorithm adopted in their implementation was given in the publication, although those settings and parameters may greatly affect iFeedback’s issue detection capabilities.

4.1 Subject Systems

In our experiments, we applied SKYNET to three industry-level online service systems. Table 1 summarizes the basic information about the systems. For each system, the table gives its ID, a brief description, its number of monthly active users (MAUs) in millions, and the average number of user feedback items received per day for the system. System S1 is an online video-sharing social media platform, system S2 is an online video editing system, and system S3 is an online beauty camera platform. The subjects include systems of different types for different users, with different magnitudes of MAUs, and receiving different amounts of user feedback. The diversity in the subject systems helps to ensure that the experiments are representative of SKYNET’s behavior in different situations.

4.2 Model Training

Since all three subject systems mainly target Chinese users, we configured SKYNET to utilize a pre-trained ALBERT model [1], the DSG embedding corpora [7], and the Jieba text segmentation library [5] for processing texts in Chinese. Meanwhile, we configured SKYNET to utilize the texts posted on Weibo³, one of the biggest social media platforms in China, for issue detection in the experiments.

For each system, we utilized historical user feedback with labels manually assigned by the system developers over a one-month period to fine-tune the ALBERT-Tiny model and to train the cascading classification model as a whole. To prepare the hierarchical label system, first, we invited the system developers to decide which labels associated with negative user experience reporting feedback should be retained as the bottom layer labels. Then, following the principles described in Section 3.1, the developers were asked to group and summarize the bottom layer labels to form the intermediate and top layer labels. Finally, all the other labels indicating negative user experiences were converted to “Unknown”, and the remaining labels were converted to “Non-negative”. In this way, we prepared for each online service a hierarchical label system and a large number of user feedback associated with those labels. For each constructed hierarchical label system, Table 1 gives the numbers of labels at its three different layers.

Afterward, we followed the standard practice [34] to tune the hyperparameters to be used with the classification and BiLSTM models. Particularly, for each service system, we selected via random search a group of 10 hyperparameters that enables the classification model to correctly label the most historical user feedback texts, and then we looked for values adjacent to these hyperparameters via grid search [34] that produced the highest number of correct labels and used the values for the classification model in our experiments. The BiLSTM model was trained through stochastic gradient descent [37] on the time series data derived from the given historical feedback data. For example, for the experiments on service system S1, the cascading classification model used the following non-default hyperparameters: `batch_size=24`; `dropout=0.1`; `learning_rate=2e-5`; `warm_up_proportion=0.1`; `max_epoch=10`, while the BiLSTM model used the following non-default hyperparameters: `dropout=0.1`; `max_epoch=50`; `sequence_len=50`; `learning_rate=0.1`; `batch_size=24`.

4.3 Experimental Setup

We applied SKYNET to detect issues in each subject system based on historical data collected over a ten-month period of time. Each detected issue was checked manually by operators and developers of the systems to confirm whether it indicates a real problem that needs to be handled. Moreover, the operators and developers also assessed the severity of each issue based on the functionalities it may impact, the costs it may incur, and the extent to which users’ experience may be jeopardized. An issue is called a *severe* issue if its impact in at least one of those aspects is substantial.

³ <https://www.weibo.com>

To answer RQ1, we collected all the issues reported by SKYNET for the subject systems as well as the results of manual inspections on the issues. Following the practice in previous work [45], we measure the effectiveness of SKYNET in terms of the *precision* and *recall* of the issue detection results produced by the tool. In particular, the precision is calculated as the percentage of real issues in all the detected issues, i.e., N_c^i/N_d^i , where N_c^i and N_d^i are the numbers of issues confirmed by developers and detected by SKYNET, respectively; The recall is calculated as the ratio of detected severe issues to all the severe issues recorded for the whole experiment period, i.e., N_d^s/N_r^s , where N_d^s and N_r^s are the numbers of severe issues detected by SKYNET and recorded by developers, respectively. Note that metric recall concerns only severe issues in the system because severe issues will be reported eventually due to their high impact even if SKYNET fails to detect them, while there is no practical way for us to find out the exact total number of real issues in those systems.

To answer RQ2, we ran SKYNET two more times on all the user feedback data and the social media posts to detect issues for the systems, the first time with component C_p being disabled and the second time with both components C_p and C_u being disabled. Then, we compared the issue detection results from the three runs in the number of issues detected as well as the precision and recall of the corresponding results.

4.4 Experimental Results

In this section, we report on the results produced in the experiments and answer the research questions.

RQ1: Effectiveness Table 2 lists the basic information about the issue detection results SKYNET produced on the systems. For each system, the table lists its system ID, the numbers of issues detected by SKYNET and confirmed by developers, the numbers of severe issues detected by SKYNET and recorded by developers, and the precision (PREC) and recall (RECA) achieved accordingly.

SKYNET detected 2790 issues in total, 2595 of them were manually confirmed to be true issues, achieving a precision of 93.0%. As for severe issues, developers recorded in total 62 cases for the three systems in ten months, and 58 of them were detected by SKYNET, achieving a recall of 93.5%. In comparison, iFeedback [45] was able to achieve 76.2% and 93.2% for precision and recall, respectively, in its evaluation. SKYNET managed to significantly outperform iFeedback in terms of precision while slightly improving the recall. Such results suggest that SKYNET is both effective and accurate in issue detection.

To understand the reasons for SKYNET’s ineffectiveness, we manually inspected all four severe issues that were missed. Three of the four severe issues were missed due to minor fluctuations in the number of associated user feedback. For instance, one severe issue that SKYNET missed occurred during AB-testing [40] of a service system. Since only a small number of users were involved in the AB-test, while the issue seriously damaged the user experience of the system,

Table 2: Issue detection results produced by SKYNET on the subject systems.

SID	ISSUE		SEVERE ISSUE		PREC	RECA
	DETECTED	CONFIRMED	DETECTED	RECORDED		
S1	2003	1895	51	54	94.6%	94.4%
S2	507	452	7	8	89.2%	87.5%
S3	280	248	0	0	88.6%	-
Overall	2790	2595	58	62	93.0%	93.5%

Table 3: Usefulness of SKYNET’s individual components for issue detection.

SID	SIR	C_k					$C_k + C_u$					SKYNET ($C_k + C_u + C_p$)				
		N_d^i	N_c^i	N_d^s	P	R	N_d^i	N_c^i	N_d^s	P	R	N_d^i	N_c^i	N_d^s	P	R
S1	54	1975	1870	28	94.7%	51.9%	1997	1889	45	94.6%	83.3%	2003	1895	51	94.6%	94.4%
S2	8	497	444	5	89.3%	62.5%	507	452	7	89.2%	87.5%	507	452	7	89.2%	87.5%
S3	0	277	246	0	88.8%	-	280	248	0	88.6%	-	280	248	0	88.6%	-
Overall	62	2749	2560	33	93.1%	53.2%	2784	2589	52	93.0%	83.9%	2790	2595	58	93.0%	93.5%

the total number of users affected was relatively small, compared with the number of users that routinely access the service provided by the system. Hence, no alert was triggered. The severe issue could have been detected if SKYNET predicts the threshold frequency of issue-reporting feedback texts as a ratio to the total number of users with access to the relevant system feature. SKYNET missed the other severe issue of a previously unknown type due to the imprecise clustering of feedback texts. Since various users’ descriptions of the issue were quite different, SKYNET’s unsupervised model was not able to group all the user feedback reporting the same issue into a cluster. This is not completely unexpected since, although we have considered both the lexical and semantic characteristics of feedback texts in their embedding, it is not a perfect solution yet. We plan to devise more powerful embedding and clustering techniques to facilitate the detection of issues of unknown types in the future.

SKYNET was effective and accurate in detecting issues for large-scale online service systems. 93.0% of the issues detected by SKYNET reflect real problems that demand manual inspection. 93.5% of the severe issues recorded for the systems were detected by SKYNET.

RQ2: Usefulness of Component Mechanisms Table 3 shows the results produced by SKYNET with various components being disabled in issue detection. For each system identified by its SID, the table gives the issue detection results from using just component C_k , using both components C_k and C_u , and using all three components of SKYNET. In each setting, the table lists the numbers of issues detected by the tool (N_d^i) and confirmed by developers (N_c^i), the number of severe issues detected by the tool (N_d^s), and the precision (P) and recall (R) achieved accordingly.

When C_k is the only component enabled, SKYNET was able to detect 2749 issues, among which 2560 were manually confirmed, and 33 severe issues for the

systems, achieving the overall precision and recall of 93.1% and 53.2%, respectively. To put it in perspective, that is 98.7% ($=2560/2595$) of the real issues and 56.9% ($=33/58$) of the severe issues the tool can ever detect with all its components being enabled. Such results clearly show that both cascade feedback classification and dynamic threshold prediction of SKYNET were effective in detecting issues based on user feedback. Although the recall that C_k achieved in detecting severe issues is relatively low, it is understandable since many severe issues are of previously unknown types and hence beyond the detecting capability of C_k .

Component C_u helped capture 29 ($=2589-2560$) real issues and 19 ($=52-33$) severe issues that component C_k failed to detect, which caused the precision of the overall result to drop slightly to 93.0% but helped raise the recall of the overall result to 83.9%. The drop in the result precision is understandable since C_u essentially detects issues of previously unknown types via unsupervised learning, and the results of unsupervised learning are relatively low in general. Compared with a few false positives, i.e., reported issues that were manually ruled out as they were not real issues, the 19 severe issues detected by component C_u are significantly more important for the developers. Therefore, we believe component C_u is a valuable complement to component C_k . Note that only feedback items that report negative user experiences of previously unknown types are processed by component C_u .

The issue detection results produced by components C_k and C_u also enable us to directly compare SKYNET and iFeedback’s issue detection capability solely based on user feedback. As shown in Table 3, if only having access to user feedback, or when component C_p is disabled, SKYNET was able to detect 2784 issues, among which 2589 were confirmed to be real ones and 52 were considered severe. The precision and recall achieved are therefore 93.0% and 83.9%, respectively. Recall that the precision and recall iFeedback achieved were 76.2% and 93.2%, respectively. The differences suggest that SKYNET and iFeedback make different tradeoffs between issue detection precision and recall. iFeedback is more lenient in reporting issues. On the one hand, many issues it reported turned out to be false positives; On the other hand, it managed to detect more severe issues; SKYNET is stricter in reporting issues. On the one hand, it reported fewer false positives; On the other hand, it missed a few more severe issues.

SKYNET makes up for its relatively low recall in issue detection based on user feedback by taking into account also users’ posts on social media platforms. Although component C_p only detected 6 more real issues in our experiments, all of them turned out to be severe, and missing any of these issues may have caused great damage to the company. Therefore, although this component has only slightly improved the overall recall, we consider it to be a crucial and non-dispensable part of SKYNET.

All the three components C_k , C_u , and C_p are important for SKYNET to detect (severe) issues in an effective and accurate manner.

Threat to Validity In this section, we discuss possible threats to the validity of our findings and show how we mitigate them.

Construct validity. In our evaluation, a reported issue could be manually confirmed or rejected as a real or severe issue, but different people may provide different assessments. To mitigate this threat, we directly reused the *independent* issue assessment results from the developers of the service systems.

Internal validity. SKYNET makes use of a list of parameters, including, e.g., the size of the sliding window for BiLSTM and the similarity threshold for matching social-media posts with user feedback texts. We set the parameters based on our experience in the current implementation of SKYNET. Experimental evaluation conducted on three industry-level online service systems produced very promising results, suggesting the chosen parameter values are appropriate. Having said that, we are aware that different values for the parameters may influence SKYNET’s effectiveness, and therefore we plan to conduct more experiments in the future to systematically evaluate the possible influence.

We were not able to experimentally compare SKYNET with iFeedback for reasons stated at the beginning of Section 4. As the result, we compared the two tools based on the results they produced on the subject systems in their corresponding evaluations. For the comparison to be as fair as possible, we evaluated SKYNET on service systems of similar scales from various categories of applications. Moreover, the comparison was based on common metrics precision and recall, instead of measurements like the numbers of issues and severe issues detected, which greatly depends on the experimental setup.

External validity. The subject service systems adopted in our experiments were real-world services of different scales and from different application domains. These characteristics help mitigate the risk that our evaluation overfits the subjects. In the future, on the one hand, we will continue monitoring the execution of SKYNET on existing service systems, on the other hand, we will deploy SKYNET on more service systems. We see no intrinsic limitations that would prevent SKYNET from working reliably on different online service systems.

5 Conclusions

This paper presents the SKYNET technique and tool that utilize user data gathered from multiple channels to detect issues for large-scale online service systems. The technique has been applied to detect issues for three real-world online services based on historical data gathered over a ten-month period of time. The produced results suggest that SKYNET is both effective and accurate in detecting issues and severe issues for large-scale online service systems.

6 Data Availability

The SKYNET tool has been integrated into the production issue tracking system in the first author’s company. For confidentiality reasons, neither the tool nor the multi-channel user feedback can be available for public download.

References

1. Albert pre-trained model for chinese. https://github.com/brightmart/albert_zh. Last accessed 19 May 2022.
2. Cascading classifiers - wikipedia. https://en.wikipedia.org/wiki/Cascading_classifiers. Last accessed 19 May 2022.
3. Github elasticsearch. <https://github.com/elastic/elasticsearch>. Last accessed 19 May 2022.
4. Interquartile range. https://en.wikipedia.org/wiki/Interquartile_range. Last accessed 19 May 2022.
5. Jieba - chinese text segmentation. <https://github.com/fxsjy/jieba>.
6. Okapi bm25 - wikipedia. https://en.wikipedia.org/wiki/Okapi_BM25. Last accessed 19 May 2022.
7. Tencent ai lab embedding corpora for chinese and english words and phrases. <https://ai.tencent.com/ailab/nlp/en/embedding.html>.
8. tf-idf - wikipedia. <https://en.wikipedia.org/wiki/Tf%e2%80%93idf>. Last accessed 19 May 2022.
9. S. Arora, Y. Liang, and T. Ma. A simple but tough-to-beat baseline for sentence embeddings. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.
10. D. R. Cox. The regression analysis of binary sequences. *Journal of the Royal Statistical Society: Series B (Methodological)*, 20(2):215–232, 1958.
11. J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.
12. A. Di Sorbo, S. Panichella, C. V. Alexandru, J. Shimagaki, C. A. Visaggio, G. Canfora, and H. C. Gall. What would users change in my app? summarizing app reviews for recommending software changes. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 499–510, 2016.
13. B. Fu, J. Lin, L. Li, C. Faloutsos, J. Hong, and N. Sadeh. Why people hate your app: Making sense of user feedback in a mobile app store. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1276–1284, 2013.
14. C. Gao, J. Zeng, M. R. Lyu, and I. King. Online app review analysis for identifying emerging issues. In M. Chaudron, I. Crnkovic, M. Chechik, and M. Harman, editors, *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, pages 48–58. ACM, 2018.
15. J. Gu, C. Luo, S. Qin, B. Qiao, Q. Lin, H. Zhang, Z. Li, Y. Dang, S. Cai, W. Wu, Y. Zhou, M. Chintalapati, and D. Zhang. Efficient incident identification from multi-dimensional issue reports via meta-heuristic search. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2020*, page 292–303, New York, NY, USA, 2020. Association for Computing Machinery.
16. E. Guzman, M. Ibrahim, and M. Glinz. A little bird told me: Mining tweets for requirements and software evolution. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*, pages 11–20, 2017.

17. S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, 1997.
18. C. Huang, G. Min, Y. Wu, Y. Ying, K. Pei, and Z. Xiang. Time series anomaly detection for trustworthy services in cloud computing systems. *IEEE Transactions on Big Data*, 2017.
19. T. Johann, C. Stanik, W. Maalej, et al. Safe: A simple approach for feature extraction from app descriptions and app reviews. In *2017 IEEE 25th international requirements engineering conference (RE)*, pages 21–30. IEEE, 2017.
20. Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut. ALBERT: A lite BERT for self-supervised learning of language representations. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.
21. N. Laptev, S. Amizadeh, and I. Flint. Generic and scalable framework for automated time-series anomaly detection. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1939–1947, 2015.
22. L. Li, X. Zhang, X. Zhao, H. Zhang, Y. Kang, P. Zhao, B. Qiao, S. He, P. Lee, J. Sun, F. Gao, L. Yang, Q. Lin, S. Rajmohan, Z. Xu, and D. Zhang. Fighting the fog of war: Automated incident detection for cloud systems. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, pages 131–146. USENIX Association, July 2021.
23. Z. Li, Y. Zhao, R. Liu, and D. Pei. Robust and rapid clustering of kpis for large-scale anomaly detection. In *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*, pages 1–10. IEEE, 2018.
24. Q. Lin, J.-G. Lou, H. Zhang, and D. Zhang. idice: Problem identification for emerging issues. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, pages 214–224, 2016.
25. 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*, pages 211–224, 2015.
26. M. Ma, S. Zhang, D. Pei, X. Huang, and H. Dai. Robust and rapid adaption for concept drift in software system anomaly detection. In *2018 IEEE 29th International Symposium on Software Reliability Engineering (ISSRE)*, pages 13–24. IEEE, 2018.
27. T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. In Y. Bengio and Y. LeCun, editors, *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*, 2013.
28. T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. *arXiv preprint arXiv:1310.4546*, 2013.
29. D. Pagano and W. Maalej. User feedback in the appstore: An empirical study. In *2013 21st IEEE international requirements engineering conference (RE)*, pages 125–134. IEEE, 2013.
30. F. Palomba, M. Linares-Vásquez, G. Bavota, R. Oliveto, M. Di Penta, D. Poshyvanyk, and A. De Lucia. User reviews matter! tracking crowdsourced reviews to support evolution of successful apps. In *2015 IEEE international conference on software maintenance and evolution (ICSME)*, pages 291–300. IEEE, 2015.
31. S. K. Panda and P. K. Jana. Efficient task scheduling algorithms for heterogeneous multi-cloud environment. *The Journal of Supercomputing*, 71(4):1505–1533, 2015.

32. S. Patro and K. K. Sahu. Normalization: A preprocessing stage. *arXiv preprint arXiv:1503.06462*, 2015.
33. M. Raginsky, R. M. Willett, C. Horn, J. Silva, and R. F. Marcia. Sequential anomaly detection in the presence of noise and limited feedback. *IEEE Transactions on Information Theory*, 58(8):5544–5562, 2012.
34. K. Ramasubramanian and J. Moolayil. *Applied Supervised Learning with R: Use machine learning libraries of R to build models that solve business problems and predict future trends*. Packt Publishing, 2019.
35. M. Saberian and N. Vasconcelos. Boosting algorithms for detector cascade learning. *Journal of Machine Learning Research*, 15:2569–2605, 2014.
36. R. Sathya, A. Abraham, et al. Comparison of supervised and unsupervised learning algorithms for pattern classification. *International Journal of Advanced Research in Artificial Intelligence*, 2(2):34–38, 2013.
37. S. Sra, S. Nowozin, and S. J. Wright. *Optimization for machine learning*. Mit Press, 2012.
38. C. Stanik, T. Pietz, and W. Maalej. Unsupervised topic discovery in user comments. In *2021 IEEE 29th International Requirements Engineering Conference (RE)*, pages 150–161. IEEE, 2021.
39. Y. Sun, Y. Zhao, Y. Su, D. Liu, X. Nie, Y. Meng, S. Cheng, D. Pei, S. Zhang, X. Qu, et al. Hotspot: Anomaly localization for additive kpis with multi-dimensional attributes. *IEEE Access*, 6:10909–10923, 2018.
40. D. Tang, A. Agarwal, D. O’Brien, and M. Meyer. Overlapping experiment infrastructure: More, better, faster experimentation. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 17–26, 2010.
41. L. Villarroel, G. Bavota, B. Russo, R. Oliveto, and M. Di Penta. Release planning of mobile apps based on user reviews. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, pages 14–24. IEEE, 2016.
42. P. M. Vu, H. V. Pham, T. T. Nguyen, and T. T. Nguyen. Phrase-based extraction of user opinions in mobile app reviews. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, pages 726–731, 2016.
43. G. Williams and A. Mahmoud. Mining twitter feeds for software user requirements. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*, pages 1–10. IEEE, 2017.
44. H. Xu, W. Chen, N. Zhao, Z. Li, J. Bu, Z. Li, Y. Liu, Y. Zhao, D. Pei, Y. Feng, et al. Unsupervised anomaly detection via variational auto-encoder for seasonal kpis in web applications. In *Proceedings of the 2018 World Wide Web Conference*, pages 187–196, 2018.
45. W. Zheng, H. Lu, Y. Zhou, J. Liang, H. Zheng, and Y. Deng. ifeedback: exploiting user feedback for real-time issue detection in large-scale online service systems. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 352–363. IEEE, 2019.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

