LogInsights: Understanding and Extracting Information from Logs for Fault Classification at run-time

Anonymous ACL submission

Abstract

Software monitoring is the most critical part in any software management life cycle. One of the ways to detect the health of the program and the software is to monitor the logs efficiently. In this paper, we describe a method to process a stream of logs for identifying any fault being mentioned in the log at runtime. At first, we extract meaningful features for detecting the erroneous ones from the stream of logs. Next, we categorize the erroneous logs into the pre-defined categories of commonly occurring faults, using the proposed two-step framework. We propose efficient, fast and intelligent rule-based systems with the domain knowledge being incorporated using the word embedding model. We have built a domain specific corpus and trained a word embedding model for this purpose. The methods described here have shown improved results in the existing product pipeline. Experiments on logs obtained from various applications also show the efficacy of our proposed method.

1 Introduction

Organizations are now in the path of rapid digitization, with thousands of applications, services, hardware systems and microservices running in a hybrid cloud environment. In today's world of hybrid cloud, understanding why a service fails and what incident remediation steps to perform that would result in minimal downtime are extremely challenging tasks. One of the key roles of an IT operations engineer is to support these applications and keep the services running. At present, an engineer manually analyses the IT operational data and uses a large number of tools to diagnose the root cause(s) of a failure in order to decide the most effective remediation action. Usually, these operations result in longer mean times to detect and longer problem resolution times.

An important IT operational data - logs generated from multiple data sources, can provide key insights to help detect the status or the problem(s) of various components. However, due to the ever-rising volume and variety of log data, the main challenge before an operations team is how to effectively use and analyze it. To meet this challenge, one needs to mine the data, discover knowledge, and use the insights so gained in failure management tasks while significantly reducing manual effort and visual overload on IT operators.

One of the needs for log processing is to have time-efficient modules at the beginning of the AIOPS pipeline. As the rate at which log streams come is very fast (around 10k per sec), it is essential that the processing time needs to be very fast, to avoid any time-lapse. Also, the module needs to be generalized enough to be efficient of various log formats from different sources. We go beyond the traditional rule-based methods, where error clues are either extracted using regex pattern matching or are dictionary-based built manually (Zou et al., 2014). We aim to approximate a sophisticated probabilistic model by intelligent unsupervised methods using:

(i) automatic domain-specific dictionary building,
(ii) focusing on important sub-text/features from log message
(iii) using a two-step fault categorization module for meeting the need to 'time to value'.

The domain knowledge is being incorporated in the unsupervised approach using the word embedding model trained on IT corpus. Most of the time-efficient methods of log category detection in event logs rely on simple dictionary-based matching or rules-based systems. Our proposed method approximates a probabilistic model in a time-efficient manner using a domain-specific word embedding model in a two-step process.

This paper describes a method of identifying the type of faults in an erroneous log. As the log stream comes in, first an error detection module identifies the erroneous logs from the input stream, for which we identify the fault. This two-step process helps us to design a time-efficient yet effective method for fault categorization. The proposed method has been built on top of an existing system of log analysis. We have successfully improved the error detection module and proposed a new domain-specific, fast and efficient fault categorization module, which is being discussed in the rest of the paper.

2 Prior Work

On log curation, there has been work (He et al., 2020a) on curating log datasets from real-world systems including Hadoop, HDFS, Openstack, etc. The primary goal was to train and evaluate log parsing schemes. In contrast, our approach required a corpora that enables learning of higher level semantics in the technical vernacular, that log datasets simply do not have.

On the topic of log parsing and template learning for there has been a garden variety of approaches both rule-based (Hansen and Atkins, 1993; Prewett, 2003; Rouillard, 2004), and pattern mining algorithms such as LogCluster (Vaarandi and Pihelgas, 2015), SLCT (Vaarandi, 2003), LKE (Fu et al., 2009), DRAIN (He et al., 2017a), SPELL (Du and Li, 2019), IPLoM (Makanju et al., 2009), LenMA (Shima, 2016), and others (Nagappan et al., 2009). We propose here to avoid learning templates, and simply use off-the-shell dependency parsing and tagging (see Honnibal et al., 2020) applied only on a small set of identified loglines.

There are few works on identifying fault categories of event logs. (Zou et al., 2014) proposes to consider only the invariant tokens in the log lines which are identified using templatization. The fault categories are detected by using a Fault-Keyword matrix which denotes affinity of a token/keyword with a particular fault category. This is done by clustering the loglines together and then calculating the affinity using tf-idf. This method is definitely limited to the vocabulary of the log dataset and is not very easily extendable. Also, methods like clustering and tf-idf based weight calculation may not really capture the semantics of the loglines, which is one of the main focus in our work.

On anomaly detection there are various approaches such as the following, spanning unsupervised (Ramaswamy et al., 2000; Dickinson et al., 2001; Lou et al., 2010), one-class supervised (Mirgorodskiy et al., 2006), and supervised (Yuan et al., 2006; Xu et al., 2008). In particular more recent techniques improve by relying on sophisticated NLP techniques that use embeddings and probabilities models such as neural nets (Wang et al., 2020; Du et al., 2017). We propose here to simplify for production settings, by employing unsupervised (domain-specific) embeddings, in ways that approximate probabilistic models, however here involving limited supervision in updating the models. This allows us to go a step beyond anomaly detection to fault categorization, where we employ a combination of dictionary specific and embedding techniques that meet production latency and throughput demands.

3 Proposed Method

Probabilistic models based on sophisticated NLP techniques (see Wang et al., 2020; Du et al., 2017) are state-of-the-art, but face issues in the production demands of a high-throughput, low-latency application of log analysis. Given a training set of log data $\{x_i\}_{i=1}^{n}$, where n is the number of training examples, erroneous log detection and fault categorization, involve learning a function $f(\cdot, \theta)$, that optimizes parameters θ such that predictions $y_e, y_f = f(\cdot, \theta)$ corresponding to erroneous log y_e, and fault category prediction y_f, perform well against some loss measure; here y_e is binary, and whenever $y_e = 1$ then y_f lies in some set of predefined fault categories C, whenever $y_e = 0$ then y_f is irrelevant and left undefined. Probabilistic models typically employ a nonlinear $f(\cdot, \theta)$ that is trained in a supervised manner. Here, we propose to approximate such an f that can be constructed in ways more amenable to production settings, where the parameters $\theta = (D_n, D_s, \{D_i\}_{i \in C}, \Phi)$ involve i) dictionaries $D_\mathcal{S}$ (existing symptom dictionary), $D_\mathcal{N}$ (negative sentiment dictionary) and $\{D_i\}_{i \in C}$ (one for each fault category) that are preconstructed with minimal human intervention and ii) a set of word embeddings Φ. We propose a two-step approach to construct y_e and y_f:

- use an existing symptom dictionary $D_\mathcal{S}$ and proposed sentiment dictionary $D_\mathcal{N}$ obtained in an unsupervised manner to predict y_e.

- if $y_e = 1$, approximate the manner in which probabilistic models predict the fault category y_f, by a novel combination of dependency tags features, dictionaries $\{D_i\}_{i \in C}$, and the embeddings Φ. In particular, each dictionary

D_i will be constructed in an unsupervised manner. If $y_e = 0$, there is no need for identifying fault.

We use domain-specific pre-trained embeddings Φ extensively trained in an unsupervised manner, for (i) domain-specific dictionary building (offline process) and (ii) unsupervised multi-label fault categorization, that allows better performance over technical vernacular; this gets rid of the need to retrain for individual log datasets.

The flowchart in Figure 1 shows the proposed method. The rest of the modules are explained in detail in the following sub-sections. The framework is currently being absorbed in the existing product pipeline for improving the log-anomaly detection pipeline in a phased manner.

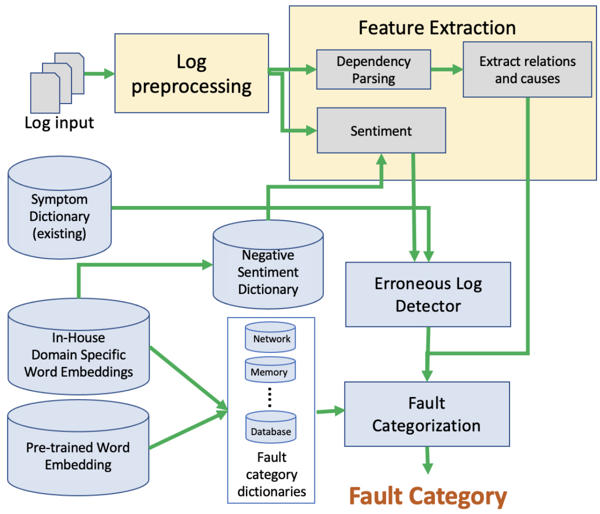

Figure 1: Proposed framework for fault categorization in logs.

3.1 Domain Specific Word Embedding Model

We use domain specific word embedding to build domain-specific dictionaries with minimum human intervention and to calculate similarity-based multi-label fault classifiers. We require embeddings that are adapted for our domain of technical vernacular that appears in log data. This is because, there is a need to deal with words that appear in the regular language that have different technical meanings such as block, application, server, web, etc. There are technical jargons that are not common in the general English corpus. Also, words like “bug" can have a different meaning in a technical domain (faulty in the technical domain, insect in general English). In order to understand the semantic meaning of the given sentence/part of a sentence, we rely on a domain-specific word embedding model.

Our approach is to go beyond log data to obtain a technical corpus that captures semantic relationships between technical terms. Log data is inherently generated from templates, and thus may lack the semantic diversity required for corpora when training embeddings. We use three types of word embedding model in our framework: (i) Glove model trained on IT-related corpus, (ii) Fasttext model trained on log messages and (iii) pre-trained Fasttext model trained on common-crawl and Wikipedia.

3.1.1 Glove Word Embeddings

Assembling the Corpora - Table 1 summarizes the billion-word corpora used for model training, focusing only on the English language in a monolingual setting. To capture the required diversity, we sourced two technical corpora - technical support articles, technical manuals, and used current events news articles for incorporating general English knowledge. The support articles were obtained from the `TechQA` dataset (Castelli et al., 2019) that contained $830K$ support pages. These support pages, or technotes, typically consists of a concise description of some problematic symptoms observed when using a particular product, along with an explanation of the fix. The technical manuals, on the other hand, incorporate text in prose form organized in topical themes (e.g., administration of a Linux server) and sub-themes (e.g., creating user accounts). These manuals were scraped from the four internet sources shown in Table 1 and account for half of the word count. The inclusion of the manuals added an additional level of diversity on top of the support pages. Finally, we used the news-wire data obtained from the Gigaword5 (gig) dataset to incorporate natural language data to ensure that the model is able to learn such semantics apart from the technical vernacular. Note that the news data is roughly about the same size as that of the support pages; this is to ensure that the technical content is dominant in our corpora.

Training - We follow popular approaches in language models learning to perform unsupervised learning. Such methods that aim to learn models that generalize well, do so without labels since labels are difficult to obtain in large quantities. Hence, we leverage models such as `glove` (Pennington et al., 2014) that can be trained over a billion words (see Table 1) without supervision.

Type	Words (mil.)	Details
Support	226.6	`TechQA` dataset (Castelli et al., 2019) of 826998 support pages
Manuals	458.5	800+ manuals from `https://access.redhat.com/documentation/en-us/` 200+ docs from `https://www.ibm.com/support/knowledgecenter/` [1] 2000+ `pdf` books from `http://www.redbooks.ibm.com/Redbooks.nsf` 200+ docs from `https://cloud.ibm.com/docs`
News	278.9	GigaWord Version 5 (gig) curated news-wire data

Table 1: Corpora for Model Training

The `glove` hyper-parameters $x_{\max}$ and α are set to default values as in (Pennington et al., 2014) along with a window size of 15, and we tune two parameters i) vocabulary size, ii) the number of train iterations.

An evaluation task inspired by *Cloze* (see (Devlin et al., 2019)) is used to tune the two hyper-parameters. A test/validation split of examples from support pages corpus (around 76856 and 77134 respectively) is constructed, each example containing a few co-located sentences. For each example, a context widow of 20 words (10 on each side) surrounding one that has been masked, along with a set of four word choices, is given to the model. The glove/BPE model determines for each of the 20 words, which of the four word choices are closer, and takes a majority vote. The task is taken to be successful if the correct word choice has the majority, and wrong otherwise. We determined a vocabulary size of 50000, and 50 and 20 iterations for embedding dimensions 50, 100, and 200, 300, respectively.

3.1.2 Fasttext Model

Another popular model for training word embeddings from scratch is Fasttext (Bojanowski et al., 2016). For our work, we have used a Fasttext model trained on log messages for having a better understanding of the semantics of the log messages. For this task, the model is being trained on logs from an IT company's Conversation Services and LogHub dataset (He et al., 2020a), using the default Fasttext parameters as mentioned in (Bojanowski et al., 2016). We used the log-anomaly detection pipeline as the evaluation task for parameter-tuning. More details of this model can be found in (Liu et al., 2020).

3.2 Feature Extraction

In this section, we describe the NLP methods that have been used for extracting meaningful features from logs as and when necessary. We consider the outputs of various log-arrgegators as input, where the logs are represented as key-value pairs, instead of considering the raw logs. The values can be of categorical, numerical or sentence-like structure, which can be processed accordingly for feature extraction. For the features proposed in this paper, we consider the values which are sentences or parts of sentences, having underlying grammar. Such values are detected, if they contain atleast two or more tokens and one verb. The features to be extracted, using NLP, are described as follows.

3.2.1 Sentiment Analysis

Log messages contain messages of missing or faulty attributes while encountering any error. For example, the log message “Unable to restart due to unknown I/O error" clearly specifies that an action has not taken place due to a certain cause. The absence of certain action/entity brings out a sense of negative sentiment in this case. We have observed that most of the erroneous log messages contain words that have a high correlation to negative sentiments. This motivates us to extract sentiment out of the log messages as a strong feature for the predictive tasks.

For purposes of analyzing negative sentiment, we propose to adopt a dictionary-based approach as opposed to full-blown ML approach; this design choice was made with the aim of processing thousands of logs per second. We build a negative sentiment dictionary for our technical domain leveraging on open-source sentiment dictionaries such as Vader (Hutto and Gilbert, 2014) and SentiWordNet (Baccianella et al., 2010). We discard words that are nouns as a candidate for sentiment dictionary; this is because in log data, negative sentiments are mostly associated with actions that most comprise of verbs, adverbs or adjectives. This is apparent in the example “block", which as a verb may be associated with negative sentiment (“blocking the gateway") whereas as a noun it is of neutral sentiment (“memory block”). We also discard any word that is out-of-vocabulary from the pre-trained embedding model; but this is not often as part of its training corpus is built from a standard English corpus. We consider any word in the vocabulary, as an entry to our negative sentiment dictionary, if any one of dictionaries (Vader and SentiWordNet) labels it as a negative sentiment word. We also add some of the words denoting negation such as “no", “n't", “not", “shouldn't" etc for its completeness. The final dictionary contains 551 words and the presence of any one of these words is considered to be of negative sentiment for the input text.

3.2.2 Relation Extraction

We extract meaningful relations between the non-copular verb present in the sentence-like structure with the corresponding subject and/or object if present. The clause must contain a verb and a subject with or without the presence of an object. On the other hand, a predicate consists of a verb along with the object associated with it. We use both clause and predicate for extraction of relevant relations as log-messages are not proper sentences. The steps used for relation extraction are as follows:

(i) Dependency parsing - This is required to identify the parts of speech and the grammatical dependencies between the words present in the input.

(ii) Verb filtration and Clause/Predicate selection - We consider only the non-copular verbs and discard any input which only contains a copular verb. We consider Subject-Verb-Object (SVO) as present in the input. In case no SVO is present, we consider a Subject-Verb (SV) which is a clause, or a Verb-Object (VO) which is a predicate.

(iii) Extension of Noun phrase - A subject or an object is a noun, which needs to be extended if required. Any adjective present just before the noun, is considered to be part of the noun phrase. Similarly, if the noun phrase has the corresponding conjunction, then the conjunction, along with its other end is also considered to be the part of the noun phrase.

(iv) Negative sentiment - We consider words depicting negative sentiment that are associated with the predicate or clause.

(v) Relation clause/phrase generation - A simple clause or phrase is generated which is in any one of the forms as (no)SVO, (no)VO, (no)SO, where “no" is optional as per the presence of negative sentiment in the input text.

3.2.3 Cause Extraction

We build on the method of cause extraction from text as described in (Sorgente et al., 2013) for log data. We consider four rules for possible cause extraction, which are as follows:

(i) Presence of Causative Verbs (Sorgente et al., 2013) - These are simple verbs denoting causal actions for logs, such as “cause", “create", “make", “generate", “trigger", “produce" and “emit".

(ii) Presence of Phrasal Verbs (Sorgente et al., 2013) - Phrases consisting of a Verb followed by a Particle or Preposition, such as: “caused by".

(iii) Presence of prepositional Adjective or Adverbial Phrases - Phrases consisting of a Adverb followed by a Preposition, such as “because of" and/or phrases consisting of a Adjective followed by a Preposition, such as “due to".

(iv) Absence of a Noun Phrase - Explicit mention of absence of a noun phrase, such as: “No file present". We look for presence of words like “no", “none" associated with a noun phrase for possible cause extraction.

3.3 Class Predictions

In this section, we briefly describe the prediction modules designed using the extracted features as described earlier. We perform two stages of prediction for fault categorization - (i) detect if a log is erroneous or not and (ii) detect the fault type in the erroneous logs only. It is possible to implement and use various sophisticated classifiers for these tasks. However, one of the most important requirements of these models is to be extremely fast, which can process thousands of streaming logs in seconds. The proposed classifiers aim to approximate time-expensive probabilistic models in an efficient way, which are described below.

3.3.1 Erroneous Log Detection

The existing erroneous log detection module uses a pre-built symptom dictionary for detecting erroneous logs. The symptom dictionary has been built using operation engineers knowledge of faulty logs, as described in (Ray et al., 2020). If any word from the symptom dictionary is present in the input log message, it is classified as an erroneous log. However, it was observed that this produced a lot of false-positive alarms. We propose a more stringent rule, using both the existing symptom dictionary and the negative sentiment dictionary and modify the error classifier as:

$$y_e(t) = \begin{cases} 1, & \text{if } (w(t) \cap w(\mathcal{S}) \neq \emptyset) \\ & \wedge (w(t) \cap w(\mathcal{N}) \neq \emptyset). \\ 0, & \text{otherwise.} \end{cases}$$

where, y_e is the class label of error classifier, $w(.)$ denotes all the words/tokens of the input as a set and t, $\mathcal{S}$, $\mathcal{N}$ represent the input text, the symptom dictionary and the proposed negative sentiment dictionary respectively. It is to be noted that some tokens, such as "exception", are common to both the symptom and negative sentiment dictionaries.

3.3.2 Fault Categorization

In the next step, we detect the fault category of the erroneous logs. We consider 8 fault categories (database, disk, file, memory, network, protocol, storage, others), as proposed in (Zou et al., 2014). We propose an adaptive rule-based classification approach in order to make the process time-efficient. We automatically build dictionaries corresponding to each of the categories (except for the category "others") using the word embedding model. For a particular category name, we consider the nearest nouns which are within a particular distance threshold from the category name. For example, for the category "database", some of the tokens in the dictionary are "jdbc", "sql", "knowledgebase", "query" etc. This is a fast approximation of the Parzen window classifier (window size determined by k-fold cross-validation) in the embedded space. At run-time, for a log message, we first extract the relations and causal phrases as described in subsections 3.2.2 and 3.2.3, as we want to concentrate only on the meaningful parts of the log messages. Next, we check if any words from the extracted relation and/or causal phrases exist in any one of the category dictionaries.

For a logline input x, the predicted fault categories $y_f = y_f(x)$ is a subset of labels in C obtained as:

$$m(x) = \min_{a \in D_C, b \in w(x) \cap D_\Psi} |\Psi(a) - \Psi(b)|_2 \quad (1)$$

$$y_f(x) = \{i \in C : \mathtt{sigm}(m(x)) \geq T\}$$

where if the minimum (1) is well-defined then $m(x) = m(x; D_C, \Psi)$ denotes the minimum distance between word pairs a, b, where a is a word from input x with a valid entry in dictionary D_Ψ, and b is an entry from the dictionary D_c, and $\mathtt{sigm}(u) = (1 - \exp(-u))^{-1}$ is the sigmoid function for unrestricted u, and T is a threshold that is chosen using the test set. In the case where for x we have $\in w(x) \cap D_\Psi$ to be empty and (1) is not well-defined (when none of the words in the input is from the word embedding vocabulary), then we set $y_f(x) = \{\mathtt{others}\}$ to contain only a single $\mathtt{others}$ label (distinguished from any label in C). Recall that fault categories are only predicted when erroneous logs are predicted (i.e., only when $y_e(x) = 1$). We expect that under usual operating conditions $y_e(x) = 1$ is only for a reasonably small set of log-lines, hence the operation (1) though expensive, only needs to be done for a sparse amount of time.

Figure 2 shows the intermediate steps of fault categorization for an example. It shows, how after pre-processing, the input is being detected as erroneous log using symptom and negative sentiment dictionary. For fault categorization, the extracted symptom and cause phrase are being showed along with the final fault category.

4 Results

We evaluate the proposed framework on four different log dataset: (1) Private log data from an IT company's Conversation services (CS) ($\sim 900K$ instances), (2) Socshop (8648 instances), (3) HDFS ($\sim 610K$ instances) and (4) QOTD ($\sim 180K$ instances), where QOTD and Socshop are simulated datasets and HDFS is an open-source dataset (He et al., 2020b).

To build a labeled test set for experimentation, we used the Drain template miner (He et al., 2017b) on the log messages. We mined 41 logline templates for HDFS, 64 templates for QOTD, 74 templates for Socshop and 51 templates for the CS dataset. The reduced test set contained only the log templates, allowing us to hand annotate the samples for the tasks of error detection and fault categorization using the suggestions from operation engineers. For efficient feature extraction, we first performed efficient pre-processing using an in-house parser. Oftentimes, a nested JSON or XML is present as a part of the string as a value in the key-value pairs. We parsed these nested structures and created a list of key-value pairs, considering only the values which are strings having an underlying grammar.

4.1 Erroneous Log Detection

We compare the erroneous log detection module for three different variations: (i) Method 1 - using only the symptom dictionary as implemented in the baseline; (ii) Method 2 - using the proposed method

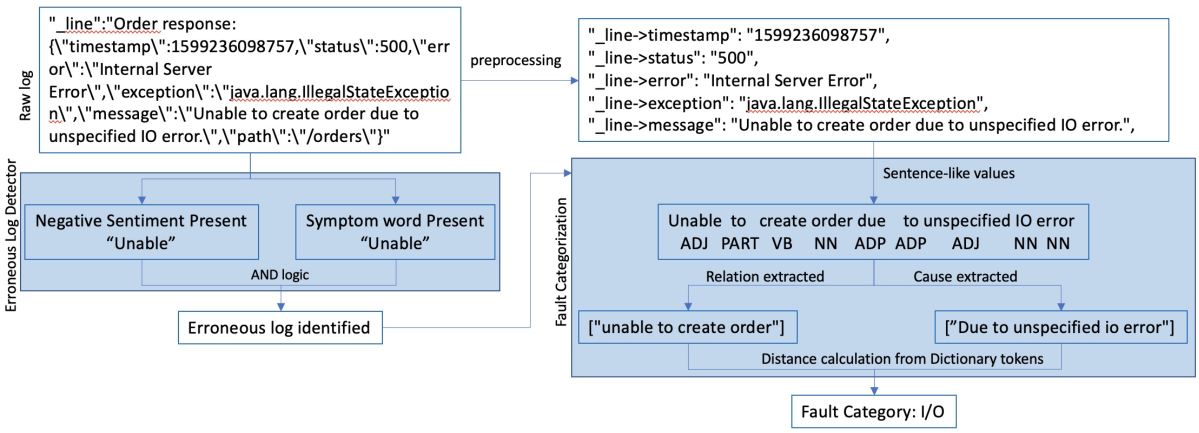

Figure 2: Example showing the different steps of fault categorization.

where both symptom dictionary and negative sentiment dictionary are being used; and (iii) Method 3 - using probabilistic sentiment detector along with the symptom dictionary. Table 2 shows the performance comparison of the different variations on the four log datasets for the task of errorneous log detection (ED). The first three rows show the F1 score, while the next three rows show the average time taken, in micro-seconds, for each log, using the three methods. The last two rows show how the performance of the existing Log Anomaly Detection (LAD) module (Liu et al., 2021) improves as it considers the output of the improved error detection module.

		CS	SocShop	HDFS	QOTD
ED F1	M1	0.84	0.67	0.45	1.00
	M2	0.89	0.77	0.78	0.88
	M3	0.93	0.89	0.88	0.88
ED μsec	M1	356	127	49	47
	M2	1935	220	254	216
	M3	9145	1903	613	595
LAD F1	M1	0.45	0.40	0.32	0.93
	M2	0.48	0.80	0.33	0.99

Table 2: Performance of error detection and Log anomaly detection using various methods.

It is evident from the results that the performance of the log error detector improved by a significant margin as we included the sentiment as a feature. The improvements in the F1 score are mainly due to the reduction in the false-positive numbers for all the four datasets. As shown, using a sophisticated sentiment analyser from the Textblob toolkit (tex) outperforms the other methods, but the processing time also increases. As a result, a probabilistic sentiment analyzer could not be implemented in the existing pipeline for the downstream task of log anomaly detection. The comparison of the proposed method with the baseline (method 1) shows the effectiveness of sentiment analyzer, as a feature, for erroneous log detector.

As we prefer a dictionary-based sentiment analyzer, we see some limitations due to which the error detection is incorrect. For example, in the log message “Receiving empty packet for block blk", the sentiment is non-negative as there is no word “empty" in the dictionary. Currently, we are not manually adding any tokens in the negative sentiment dictionary to keep it generic enough. Also, adding specific tokens in the dictionary does not always guarantee a better F1 score and may increase the number of false positives. In another scenario, there are arbitrary log messages, which lead to false positives, such as “We were not paid to sell this sock. It’s just a bit geeky". These ablations can be further reduced by adopting a better tokenizer during the pre-processing time and updating the dictionary automatically as more log data are being encountered.

4.2 Fault Categorization

We categorize the erroneous logs into one of the eight fault classes as described earlier. We show the effectiveness of the fault categorization in Table 3. As shown in the table, using Domain-Specific (DS) embeddings trained on technical and log corpus achieve better F1 scores. Further, we show that the F1 scores improve when we consider important segments (relations and causes) of the log messages. This is due to the reduction of false-positive samples, by weeding out segments of the log message which are neither a part of a relation nor cause. For log messages that do not contain relations and/or causes, we consider the original log message for the task of fault categorization.

	CS	SocShop	HDFS	QOTD
Pretrained Embd.	0.81	0.82	0.73	0.65
DS Embd.	0.84	0.85	0.76	0.66
DS Embd. (rel. & cause)	0.89	0.87	0.88	0.9

Table 3: Performance of fault classification using pretrained and domain specific embeddings (with & w/o feature extraction)

The predicated categories on a few log messages are shown in Figure 3. We highlight the words in each of the log messages which was responsible for the categorization task in red color. We also show a few examples where the fault category prediction was wrong (Figure 4). On further analysis, we find that the error mainly happens because the fault from the previous modules propagates into the fault classification module. Figure 4 shows examples of limitations of the dictionary approach for both fault categorization and error detection and improper feature extraction. Another reason for misclassification happens when two different fault categories are highly correlated with each other. For example, often "authentication" is needed for accessing "databases" and related terms of these two categories occur in close proximity. Thus words related to these two categories will be close to each other in the embedded space, which causes the increase of misclassification rate in fault categorization.

[orders,,,] 7 --- [xxxx] o.a.c.c.C.[.[.[/].[dispatcherServlet] : Servlet.service() for servlet [dispatcherServlet] in context with path [] threw exception [Request processing failed; nested exception is java.lang.IllegalStateException: Unable to create order due to unspecified IO error.] with root cause - **['i/o']**
{"t":{"$date":"2020-09-04T16:55:48.746+00:00"}, "c":"NETWORK", "id":xxx, "ctx":"connxxx","msg":"Error receiving request from client. Ending connection from remote","attr":{"error":{"codeName":"ProtocolError","errmsg":"Client sent an HTTP request over a native MongoDB connection"},"remote":"x.x.x.x:x","connectionId":xxx}} - **['database', 'network', 'protocol']**
{"timeMillis":1578755700979,"thread":"Threadx","level":"INFO","loggerName":"com.ibm.watson.voyager.syncer.workers.PriorityRequestWorker","message":"Thread timed out waiting for Redis queue, getting new Redis client: throttlePrefix: xyz, queueName:xyz_priority, timeout(s):60", "endOfBatch":false, "loggerFqcn":"org.apache.xxx.AbstractLogger", "contextMap":{}, "threadId":25, "threadPriority":5} - **['memory']**
{'message': 'receiving empty packet for block blk_-12345', 'timestamp': 1577838860000, 'instance_id': 'hdfs', 'entities': {'block_id': 'blk_-12345'}, 'features': [], 'meta_features': [{'type': 'windowmeta', 'obj_value': {'start': 1577838860000, 'end': 1577838870000}}], 'application_group_id': 1, 'application_id': 1} - **['network, disk, protocol']**
{'message': 'exception in receiveblock for block blk_-12345 java.io.ioexception: connection reset by peer', 'timestamp': 1577836840000, 'instance_id': 'hdfs', 'entities': {'block_id': 'blk_-12345'}, 'features': [], 'meta_features': [{'type': 'windowmeta', 'obj_value': {'start': 1577836840000, 'end': 1577836850000}}], 'application_group_id': 1, 'application_id': 1} - **['i/o']**

Figure 3: Logline examples with predicted Fault Categories.

5 Conclusion

In this paper, we propose a novel method of fault categorization in event logs in a time-efficient way. We first detect the erroneous ones from the stream

False Negative	
Word absent in Negative Dictionary	Execution of Rabbit message listener failed.
Entity not being extracted	addStoredBlock request received for blk_-123 on 0.0.0.0:0 size 123 But it does not belong to any file
Word absent in Symptom Dictionary	PendingReplicationMonitor timed out block blk_-123
False Positive	
Dictionary related error	Attempting to delete cart for user: 123 - ['authentication', 'database']
Nearest neighbors via embeddings	

Figure 4: Ablation study: Fault Categories from Log Lines

of logs by analyzing the sentiment of the log message. In the next step, we detect the fault category of the erroneous logs by extracting important segments of the log message and using the pre-built fault category dictionaries. The domain knowledge has been incorporated on all the modules with the efficient use of the word embedding model, trained on technical support documents. We also show that the proposed method significantly improves the existing Log-Anomaly detection pipeline. In future work, we plan to explore and improve the performance by taking into account the probability of a token belonging to a particular category dictionary. This may lead to improvement as a particular token can exist in multiple dictionaries. Use of improved tokenizer or embedding model will also enable us to handle words which are out of the vocabulary. We also plan to investigate approaches and ways to extend the framework to new fault categories based on client needs or new log types being encountered. This proposed framework is the base that can be upgraded to a more generalizable yet robust fault categorization framework.

References

English Gigaword Fifth Edition. https://catalog.ldc.upenn.edu/LDC2011T07.

Textblob: Simplified text processing — Textblob 0.12.0 documentation. https://textblob.readthedocs.io/en/dev/.

Stefano Baccianella, Andrea Esuli, and Fabrizio Sebastiani. 2010. SentiWordNet 3.0: An enhanced lexical resource for sentiment analysis and opinion mining. In *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*. European Language Resources Association (ELRA).

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2016. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*.

Vittorio Castelli, Rishav Chakravarti, Saswati Dana, Anthony Ferritto, Radu Florian, Martin Franz, Dinesh Garg, Dinesh Khandelwal, Scott McCarley, Mike McCawley, Mohamed Nasr, Lin Pan, Cezar Pendus, John Pitrelli, Saurabh Pujar, Salim Roukos, Andrzej Sakrajda, Avirup Sil, Rosario Uceda-Sosa, Todd Ward, and Rong Zhang. 2019. The TechQA Dataset. *arXiv:1911.02984 [cs]*. ArXiv: 1911.02984.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv:1810.04805 [cs]*. ArXiv: 1810.04805.

William Dickinson, David Leon, and Andy Podgurski. 2001. Finding failures by cluster analysis of execution profiles. In *Proceedings of the 23rd International Conference on Software Engineering*, ICSE '01, pages 339–348, USA. IEEE Computer Society.

Min Du and Feifei Li. 2019. Spell: Online Streaming Parsing of Large Unstructured System Logs. *IEEE Transactions on Knowledge and Data Engineering*, 31(11):2213–2227. Conference Name: IEEE Transactions on Knowledge and Data Engineering.

Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. 2017. DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1285–1298, Dallas Texas USA. ACM.

Qiang Fu, Jian-guang Lou, Yi Wang, and Jiang Li. 2009. Execution Anomaly Detection in Distributed Systems through Unstructured Log Analysis.

Stephen E Hansen and E Todd Atkins. 1993. Automated System Monitoring and Notification With Swatch. page 9.

Pinjia He, Jieming Zhu, Zibin Zheng, and Michael R. Lyu. 2017a. Drain: An Online Log Parsing Approach with Fixed Depth Tree. In *2017 IEEE International Conference on Web Services (ICWS)*, pages 33–40.

Pinjia He, Jieming Zhu, Zibin Zheng, and Michael R. Lyu. 2017b. Drain: An online log parsing approach with fixed depth tree. In *2017 IEEE International Conference on Web Services (ICWS)*, pages 33–40.

Shilin He, Jieming Zhu, Pinjia He, and Michael R. Lyu. 2020a. Loghub: A Large Collection of System Log Datasets towards Automated Log Analytics. *arXiv:2008.06448 [cs]*. ArXiv: 2008.06448.

Shilin He, Jieming Zhu, Pinjia He, and Michael R. Lyu. 2020b. Loghub: A large collection of system log datasets towards automated log analytics.

Matthew Honnibal, Ines Montani, Sofie Van Landeghem, and Adriane Boyd. 2020. spaCy: Industrial-strength natural language processing in python. Doi: 10.5281/zenodo.1212303.

C. Hutto and Eric Gilbert. 2014. Vader: A parsimonious rule-based model for sentiment analysis of social media text. In *Proceedings of the International AAAI Conference on Web and Social Media*, pages 216–225.

Xiaotong Liu, Yingbei Tong, Anbang Xu, and Rama Akkiraju. 2020. Using language models to pre-train features for optimizing information technology operations management tasks. In *Service-Oriented Computing – ICSOC 2020 Workshops*.

Xiaotong Liu, Yingbei Tong, Anbang Xu, and Rama Akkiraju. 2021. Predicting information technology outages from heterogeneous logs. In *IEEE International Conference On Service-Oriented System Engineering*.

Jian-Guang Lou, Qiang Fu, Shengqi Yang, Ye Xu, and Jiang Li. 2010. Mining Invariants from Console Logs for System Problem Detection. page 14.

Adetokunbo A.O. Makanju, A. Nur Zincir-Heywood, and Evangelos E. Milios. 2009. Clustering event logs using iterative partitioning. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '09*, page 1255, Paris, France. ACM Press.

Alexander V. Mirgorodskiy, Naoya Maruyama, and Barton P. Miller. 2006. Problem Diagnosis in Large-Scale Computing Environments. In *SC '06: Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, pages 11–11.

Meiyappan Nagappan, Kesheng Wu, and Mladen A. Vouk. 2009. Efficiently Extracting Operational Profiles from Execution Logs Using Suffix Arrays. In *2009 20th International Symposium on Software Reliability Engineering*, pages 41–50, Mysuru, Karnataka, India. IEEE.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

James E Prewett. 2003. Analyzing cluster log files using Logsurfer. page 12.

Sridhar Ramaswamy, Rajeev Rastogi, and Kyuseok Shim. 2000. Efficient algorithms for mining outliers from large data sets. *ACM SIGMOD Record*, 29(2):427–438.

Anupama Ray, Pooja Aggarwal, Csaba Hadhazi, Gargi Dasgupta, and Amit M. Paradkar. 2020. Question quality improvement: Deep question understanding for incident management in technical support domain. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020*.

John P Rouillard. 2004. Real-time Log File Analysis Using the Simple Event Correlator (SEC). page 18.

Keiichi Shima. 2016. Length Matters: Clustering System Log Messages using Length of Words. *arXiv:1611.03213 [cs]*. ArXiv: 1611.03213.

Antonio Sorgente, Giuseppe Vettigli, and Francesco Mele. 2013. Automatic extraction of cause-effect relations in natural language text. In *DART@AI*IA*.

R. Vaarandi. 2003. A data clustering algorithm for mining patterns from event logs. In *Proceedings of the 3rd IEEE Workshop on IP Operations & Management (IPOM 2003) (IEEE Cat. No.03EX764)*, pages 119–126, Kansas City, MO, USA. IEEE.

Risto Vaarandi and Mauno Pihelgas. 2015. LogCluster - A data clustering and pattern mining algorithm for event logs. In *2015 11th International Conference on Network and Service Management (CNSM)*, pages 1–7.

Jin Wang, Yangning Tang, Shiming He, Changqing Zhao, Pradip Kumar Sharma, Osama Alfarraj, and Amr Tolba. 2020. LogEvent2vec: LogEvent-to-Vector Based Anomaly Detection for Large-Scale Logs in Internet of Things. *Sensors (Basel, Switzerland)*, 20(9):2451.

Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael Jordan. 2008. Mining Console Logs for Large-Scale System Problem Detection. page 6.

Chun Yuan, Ni Lao, Ji-Rong Wen, Jiwei Li, Zheng Zhang, Yi-Min Wang, and Wei-Ying Ma. 2006. Automated known problem diagnosis with event traces. In *Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems 2006*, EuroSys '06, pages 375–388, New York, NY, USA. Association for Computing Machinery.

Deqing Zou, Hao Qin, Hai Jin, Weizhong Qiang, Zongfen Han, and Xueguang Chen. 2014. Improving log-based fault diagnosis by log classification. In *Network and Parallel Computing*.