SIADS 699

Anomaly Detection in System Logs

Using Supervised and Unsupervised Machine Learning Techniques

Source: Flaticon.com

By
Manel Mahroug, Sree Valli Maganti, Thanuja P Stewart

For
SIADS 699 Summer 2023, University of Michigan [MADS]

Abstract

While logs are crucial as the primary source of information for diagnosing faults and detecting anomalies, the task is inherently challenging. In addition to their semi-structured nature, logs come in high volumes, contain interdependencies, noise and redundancy. Previous studies have explored a variety of techniques aimed at parsing, feature extraction and classification of log files. In this paper, we propose a simple, robust, and generalizable approach that uses TFIDF to vectorize log lines to be used as input for several machine learning algorithms. Our models' performance metrics show that our approach has merit as demonstrated by the high F1 score, recall, and precision metrics.

Introduction

Background

Application logs are computer generated files that contain information about all the events that occur in an application. They are used by business and technology teams to gauge the performance and security of the application and troubleshoot any problems. An anomaly in a log file is an event that does not conform to the expected pattern of the events and can be easily separated from the rest of the data points in the application logs. Anomalies can be caused by a variety of factors, which includes

- Errors in the software or hardware that is generating the log data.
- Events that occur in the system, such as a spike in traffic or a failure of a component.
- Any security breaches that happen with the application.

Motivation

Anomaly detection in log files can be an incredibly valuable tool for improving the reliability, performance, and security of the applications. Early detection of anomalies can prevent problems from escalating, prevent security breaches and avoid downtime from any data loss. The advancements made in the field of machine learning accelerated and automated the anomaly detection process compared to the time-consuming traditional techniques where the anomaly rules are more manually defined. Machine learning algorithms can automatically learn the anomaly patterns, which makes them more adaptable to changes in the data. These algorithms can be used to detect anomalies in large datasets and in real time where anomalies can occur quickly and without warning. By detecting anomalies in real time, you can take steps to mitigate them before they cause damage.

Data Source

The dataset used for the purpose of this project is the Loghub dataset which can be found in https://github.com/logpai/loghub/tree/master. This is an open source dataset and has a comprehensive collection of system logs that are freely accessible for research and academic purposes. The Loghub dataset contains a variety of different log files from distributed systems, super computer systems, operating systems, mobile systems and standalone software. The anomaly detection performed in our research used the following logs:

- BGL - These logs are from the BlueGene/L supercomputer system.
- Thunderbird - These logs are from the Thunderbird supercomputer system.

Dataset	Lines (millions)	Anomalies (thousands)	Anomaly %
BGL	4.7	348	7.40
Thunderbird	211	3248	1.54

Table 1: Dataset summary

Some of the benefits of using the Loghub dataset include:

- It is an open source dataset which is freely accessible.
- It is a large dataset and has a variety suited for machine learning.
- It is well documented with the README file provided.

Some of the challenges using this dataset include:

- The log files from different systems had different formats making it complicated to parse and pre-process in a standard way.
- The datasets had too much data for many of the models to train on which could be time consuming.

Despite the shortcomings, the dataset was still very valuable to explore using various machine learning algorithms.

Literature Review

We analyzed a diverse range of research studies that form the fundamental basis of anomaly detection. This comprehensive approach has allowed us to attain a well-rounded understanding of the task at hand. Here, we present a summary of insights extracted from a range of papers, shaping the foundations of our chosen methodology.

For log parsing, the literature indicates that there are two types of methods: clustering-based such as LKE and LogSig (He et al., 2016) and heuristic-based such as SLCT (Vaarandi,2003). Clustering-based log parsers, which inspired our approach, use various clustering techniques to group similar logs together. For heuristic-based approaches, frequent words are selected and nominated as the event candidates.

For the feature extraction phase,a few different approaches have been used. One of the most common techniques, which we adopted, is to apply windowing (fixed windows, sliding windows, and session windows) to divide a log dataset into finite chunks then generate an event count matrix where in each log sequence, the occurrence number of each log event is counted (Akidau et al., 2015).

Lastly, for the modeling stage neural network based methods, unsupervised, and to a lesser extent supervised learning methods have been used in past studies. For supervised learning, logistic regression, decision tree, and support vector machine (Zhang & Sivasubramaniam, 2008) are among the most widely used. Under the unsupervised learning methods, PCA and K-means clustering stand out as prominent techniques (Siwach & Mann, 2022), and for neural network based models LSTM and auto-encoders unmistakingly claim the top spots(Andrews et al.,2016). Though these models have reported successful outcomes, they are non-trivial; they require extensive knowledge of the various deep learning architectures and therefore present challenges

in selecting the right model for a given use case (Landauer et al., 2023). As a result, our approach, detailed in the next section, aims at presenting a simpler strategy.

Methodology

Before any supervised or unsupervised learning was performed, preprocessing was done in several stages:

Sampling

Sampling was done to make training feasible. Thunderbird, for example, has over 200 million log lines. To maintain the proportion of anomalies to normal log lines, chunks of log lines were selected randomly. Each block was 1000 lines. This was done to limit discontinuities. During sampling, the last log line of one block and the first log line of the next block are consecutive. But these log lines are not consecutive in the actual log file. By sampling larger chunks, we hope any problems from these discontinuities are minimized.

Preparsing

A simple parser per log source (BGL, Thunderbird). This parser extracted timestamps, labels, and other parts of each log line that did not vary for the entire file (Figure 1). Ideally this would be completely automated, so that new log sources could be added without manual work. We believe such automation is possible, but chose to focus our generalization efforts on processing related to machine learning.

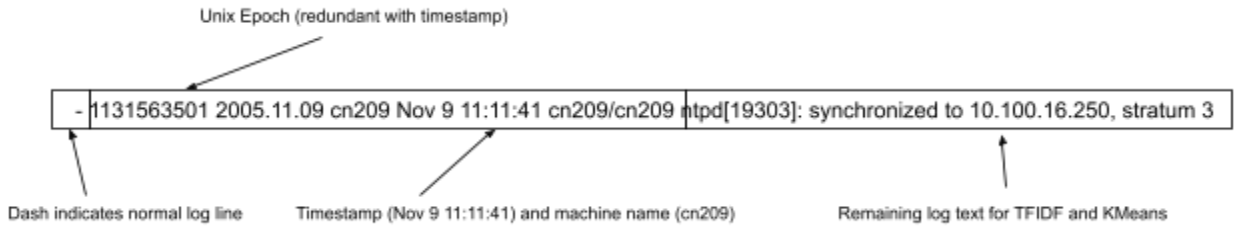

Figure 1: Example log line from Thunderbird

Clustering on TFIDF Vectors

TFIDF vectorization of the remaining log text, followed by K-means clustering of the TFIDF vectors to group them into clusters. Since the logs fit common patterns, the clustering should be informative and help learning algorithms focus on the meaning of the cluster rather than text processing. We expect clustering to work well because the log statements within an application's source code are usually limited in number and reused many times as the application runs. TFIDF vectorization removes word order information, but for the purpose of clustering, this is unlikely to be needed. Just the words used is enough to understand which type of log line it was.

The number of clusters was selected by minimizing the Davies Bouldin (DB) score while maximizing the Calinski Harabasz (CH) score. The score for each k was set to CH / DB^2. For each dataset, k was initialized to 10, then increased by 10% each iteration. The value of k was increased until k was greater than 100 and the smallest cluster had less than 20 lines.

Parameter Extraction

Within each cluster, infrequent tokens were extracted as extra features (Figure 2). The reason is that each log line type usually has a static form with parameters inserted through string formatting libraries. These are typically numbers such as “RAS KERNEL INFO **3** ddr errors(s) detected and corrected on rank **0**, symbol **16**, bit **3**”. Here each of the numbers varied by log line, and so the extra features for this log line would be 3, 0, 16, 3. In some cases, such as the time to run a process, or the number of files processed, these numbers may indicate anomalies. For example, if a process took 10 times as long to run as normal, this could indicate an abnormal state.

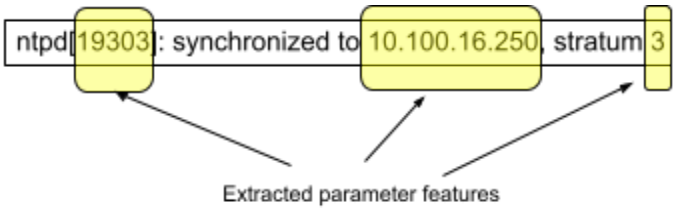

Figure 2: Log text with extracted parameters

In many other cases, these extracted parameters are not useful. For example, IP addresses, machine names, process ids, and memory addresses/offsets. For such cases, the distribution of values is expected to be uniform. Even where it’s not uniform, such as one machine name appearing more than others, this information is not useful for training. Machine names, IP addresses and so on are arbitrary and subject to change. Whatever statistical patterns may exist in the training data are not expected to persist for future logs.

Whether due to noise or sparsity, we found that the extracted parameters did not improve supervised learning. In part, this was because the performance of supervised models was already so high (see Clusters Only and Clusters w/Params in results tables below).

Parameter Filtering

A copy of the log line text column was created for filtering. As the parameters were extracted from the original clusters, they were also removed from this copy (Figure 3), but not the original text. The filtered text was then vectorized with TFIDF and clustered with K-means again. This was done to create cleaner clusters. Log lines from different clusters may share noisy tokens such as IP address and process IDs. Removing these noisy tokens improved the homogeneity with respect to ground truth labels from 95% to 99% for BGL, and from 36% to 99% for Thunderbird.

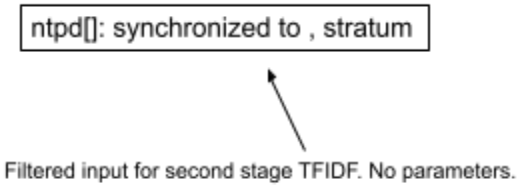

Figure 3: Log text after parameter filtering

Sliding Window

The counts of each cluster type within a sliding window (Table 2) was then used to create a new dataset. The reasoning behind this was that anomalies can be context dependent. A particular log line indicating a file

deletion may not be an anomaly normally, but if it occurs thousands of times in a row, the system may be in an anomalous state. However, the sliding window was ultimately abandoned. The BGL and Thunderbird datasets label each log line as normal or anomalous. However, using a count of clusters means that an anomalous log line will be in many consecutive windows. This proved difficult for supervised algorithms to predict whether a sliding window should be considered anomalous.

Sliding Window of Size = 3				
Original Cluster	**Clusters in Window**	**Cluster 1 Count**	**Cluster 2 Count**	**Cluster 3 Count**
3	3	0	0	1
3	3,3	0	0	2
1	3,3,1	1	0	2
3	3,1,3	1	0	2
2	1,3,2	1	1	1
2	3,2,2	0	2	1
2	2,2,2	0	3	0

Table 2: Example of a sliding window with window length = 3

We attempted to create a new label that was true if any log line within the window was anomalous, and false otherwise. But this also gave poor results with an F1 score of only 0.1028.

It also turns out that the log line type, indicated by the cluster, is enough to predict the label with over 0.99 F1 score. So for these log files at least, context doesn't matter. Then unsupervised algorithms that consider many consecutive log lines at once are at a disadvantage because all the information needed to determine if a log line is an anomaly is in the log line itself.

Modeling

All supervised and unsupervised models were trained on the following feature sets, except K-means which was only trained on TFIDF vectors:

1. *TFIDF vectors with and without filtering.* TFIDF vectors have many dimensions: 29040 for BGL and 51493 for Thunderbird on the sampled datasets. The large number of dimensions can increase training difficulty due to the curse of dimensionality. Filtering reduced the dimensionality of the dataset to 1344 for BGL and 29962 for Thunderbird.
2. *K-means clusters with and without extracted parameters.* The clusters were one-hot encoded to enable linear separation, creating 38 features for BGL and 21 for Thunderbird. The parameters added 54 features for BGL and 369 for Thunderbird. Clustering compresses each log line into one of a few dozen clusters, and the parameter features try to restore information that is lost in the clustering.
3. *Sliding window.* As described above, each feature gives the count of each cluster's occurrence within the last 100 log lines. Each parameter was averaged within each window. So the number of features is similar to one-hot encoded K-means clusters with extracted parameters. Each sliding window was labeled anomalous if any log line within the window was anomalous.

Supervised Learning

For supervised training we chose logistic regression to get a baseline. Gradient Boosted Decision Trees as well as XGBoost were also explored to see how more powerful models with non-linear decision boundaries would improve on the baseline.

For anomaly detection, the label of each log line tells whether that line is an anomaly. So supervised models can be used as a binary classifier to detect anomalies in log datasets such as BGL and Thunderbird. The explored models can also predict the probability of each line being anomalous. This allows users to apply a threshold filter to balance recall and precision. It also allows prioritizing potential anomalies based on the model's certainty.

Logistic Regression

Because of its simplicity, logistic regression is widely used for binary classification tasks and often serves as a baseline model. Logistic regression is a linear regression on the log-odds of a binary label, which is mapped to the probability of the label using the logistic function (Thanda, 2023). So the decision boundary within the input parameters must be linear for logistic regression to perform well. Based on this, the performance of a logistic regression model can tell us if the data is linearly separable.

Using the features extracted from the raw files, we trained a logistic regression model on BGL and Thunderbird datasets. During the training process we ensured the following: (1) assigning higher weights to the minority class to address the highly imbalanced nature of our datasets (2) one-hot encoding cluster labels to make them linearly separable.

Gradient Boosted Decision Trees (GBDT)

Gradient boosting is a way to combine simple models (decision trees, in our case) to progressively reduce prediction error. In almost any predictive model *M* of a complex data set, there is error. The difference of the ground truth and model predictions *y* form a new label *y - M(x)*, which is called the *residual*. Another model M' can be trained to predict the residual, then *M(x) + M'(x)* will be closer to *y* than *M(x)*. This can be repeated as many times as needed to minimize the remaining error.

In contrast to logistic regression, gradient boosted decision trees can model highly non-linear decision boundaries. Due to the sequential nature of each model correcting the previous models' errors, gradient boosting can be slow to train.

XGBoost

XGBoost is a version of GBDT that is more regularized to reduce overfitting, and parallelizable to speed up training (Chen & Guestrin, 2016). XGBoost became popular in the last decade for winning many machine learning competitions (Dmlc, n.d.). This makes it a good point of comparison to logistic regression and GBDT.

Unsupervised Learning

The unsupervised models were trained without access to the label. This is more realistic as most log data is not labeled due to its volume and the cost of manual labeling. Despite not training against labels, isolation forest and one-class SVM models still predict binary labels to indicate if a log line is anomalous. K-means on

the other hand groups related log lines and does not make any direct prediction about whether log lines are anomalous.

K-means Clustering

K-means clustering was used to group similar log lines as part of preprocessing. K-means was only run on TFIDF vectors, as running K-means on K-means output did not seem useful.

K-means starts with random clusters within the input space, then assigns each TFIDF vector to the nearest cluster. The centroid for each cluster is then recomputed based on the TFIDF vectors that are currently assigned to each cluster. The process is repeated until cluster assignments do not change. This allows many logs to be grouped together if they share most words in common, without requiring the log lines to be identical.

TFIDF vectors do not preserve word order, potentially putting unrelated log lines in the same cluster. But the log statements within an application's source code are usually limited in number and reused many times as the application runs. We did not find word order to be an issue for clustering BGL or Thunderbird.

The number of clusters was selected by minimizing the Davies Bouldin (DB) score while maximizing the Calinski Harabasz (CH) score. The score for each k was set to CH / DB^2. For each dataset, k was initialized to 10, then increased by 10% each iteration. The value of k was increased until k was greater than 100 and the smallest cluster had less than 20 log lines.

Isolation Forest

Isolation forest was designed for anomaly detection without training on labels, making this model a good fit for our project. Isolation forest creates a binary tree by repeatedly partitioning the dataset on random feature values. Log lines that are higher in the binary tree require fewer splits to be isolated from the remaining log lines. This makes them outliers or anomalies. Isolation forest has a configurable *contamination* hyperparameter that acts as a threshold for the expected proportion of anomalies.

One-Class SVM

One-class SVM was also designed for outlier or anomaly detection without labels. This model is one of the popular unsupervised learning model choices for anomaly detection due to its robustness to outliers. SGD one-class SVM was chosen due the the default 'rbf' kernel it provides which makes it sensitive to the outliers and the high speed at which this model trains on large datasets.

In one formulation, one-class SVM creates a hypersphere of minimum size to classify normal log lines. Anomalies will be log lines outside of the hypersphere. One-class SVM can detect anomalies in new data even when they were not present in the training data. Similar to the contamination parameter of isolation forest, one-class SVM has a configurable *nu* hyperparameter that sets the threshold between inliers and outliers. one-class SVM often struggles with outlier detection without careful tuning of *nu* (2.7. Novelty And Outlier Detection, n.d.).

Evaluation Strategy

Due to time limits, evaluation was performed on the sample of the log files, rather than the entire log file. While the sample was random and unbiased, it may not be fully representative of the logs within the full file.

Due to the imbalanced nature of anomaly detection datasets, unconditionally predicting the majority class could achieve high accuracy. So performance was measured with precision, recall, and F1 score.

Supervised Learning

The output of the parameter extraction was split into train and test sets. The train and test sets were split sequentially rather than randomly. This limits data leakage from future to past log events. Supervised models were trained on the training set, and then evaluated on the test set. Due to the high performance of the base supervised models, little hypertuning was performed and no validation set was necessary.

Unsupervised Learning

Unsupervised learning did not use a train and test split of the data. For isolation forest and one-class SVM, each model was fit to the entire dataset and then the predictions (inlier/outlier, normal/anomaly) were compared to the ground truth labels. For K-means, clusters were predicted from the entire dataset and homogeneity was used to compare these clusters with the ground truth. Homogeneity tells us how homogenous each cluster is. In other words, does each cluster only contain normal or anomalous log lines? With high homogeneity, it is easy to predict ground truth labels from clusters.

Discussion and Results

Supervised Results

We expected many anomalies to be dependent on context. But training TFIDF vectors against the labels gave a .9967 F1 score with logistic regression on BGL, and .9986 on Thunderbird. Each TFIDF vector represents a single log line, with no context. So context is not needed to predict anomalous log lines for these datasets.

Because context was not important for anomaly detection, supervised models performed very well with no hypertuning. So hypertuning was not explored in depth.

	TFIDF raw text	**TFIDF filtered text**	**Clusters Only**	**Clusters w/Params**	**Sliding Window**
Logistic Regression	.9549	**.9967**	.9257	.9257	.1028
GBDT	.9459	.9830	.9257	.9257	.3139
XGBoost	.9299	.9817	.6092	.6092	.1697

Table 3: BGL Test F1 Scores by model and dataset

	TFIDF raw	**TFIDF**	**Clusters**	**Clusters**	**Sliding**

	text	filtered text	Only	w/Params	Window
Logistic Regression	**1.000**	.9986	.9982	.9982	.9990
GBDT	.9991	.9986	.9982	.9982	.9990
XGBoost	.9982	.9982	.9982	.9982	.9990

Table 4: Thunderbird Test F1 Scores by model and dataset

In depth analysis was performed on the clusters with parameters dataset. In this dataset, cluster features are the cluster number prefixed with “c”. Parameter features are prefixed with “p”, followed by the cluster number and then the parameter index within the cluster.

BGL	
Feature	**Correlation with label**
c1	0.68
c32	0.47
c12	0.32
c16	0.30
c34	0.23
p-34-0	0.22
p-16-0	0.11
c9	0.06
c29	0.06
p-29-0	0.05

Thunderbird	
Feature	**Correlation with label**
c27	1.00
c31	0.08
c0	0.03
c7	0.03
c4	0.03
c17	0.03
c10	0.03
p-7-0	0.03
c16	0.02
c26	0.02

Table 5: Top 10 feature (abs) correlations with label for BGL and Thunderbird

Based on the correlations, only two parameter columns for BGL correlate significantly with anomalies, and these parameters (p-34-0 and p-16-0) belong to clusters (c34 and c16) that are themselves even more correlated with the anomalies. So the parameters do not seem to add additional information beyond the clustering. We also see that some parameters are highly correlated with anomalies, while most are not. This is because anomalies are concentrated within a few clusters. This effect is even more pronounced in Thunderbird, which has almost all anomalies concentrated within one cluster: c27.

The task for each supervised model, then, was to find the clusters that most corresponded with anomalies. This task was easy to accomplish, and the supervised models achieved exactly the same F1 score on the cluster with parameters datasets for BGL and Thunderbird. The exception was XGBoost on BGL, which is discussed below.

Logistic Regression

Logistic regression outperformed other supervised models for both datasets, achieving a .9967 F1 score for BGL (Table 3) and 1.000 for Thunderbird (Table 4) when trained on TFIDF vectors. For clusters with parameters, the F1 score reduced to .9257 (Table 3), showing that reducing log lines to clusters (even with parameters) removed important information.

	BGL(train)	BGL(test)	Thunderbird(train)	Thunderbird(test)

Precision	.9937	.8671	1.000	1.000
Recall	.9970	.9954	1.000	.9964

Table 6: Logistic Regression performance for BGL and Thunderbird datasets (clusters with params)

To gain insights into which features have the most impact on the model's predictions, we ran a feature importance analysis. Our results (Figure 4) indicate that some clusters have more predictive power than others. For thunderbird data, cluster 27, which contains almost all the dataset's anomalies, seems to have the most predictive power (Figure 4, right).

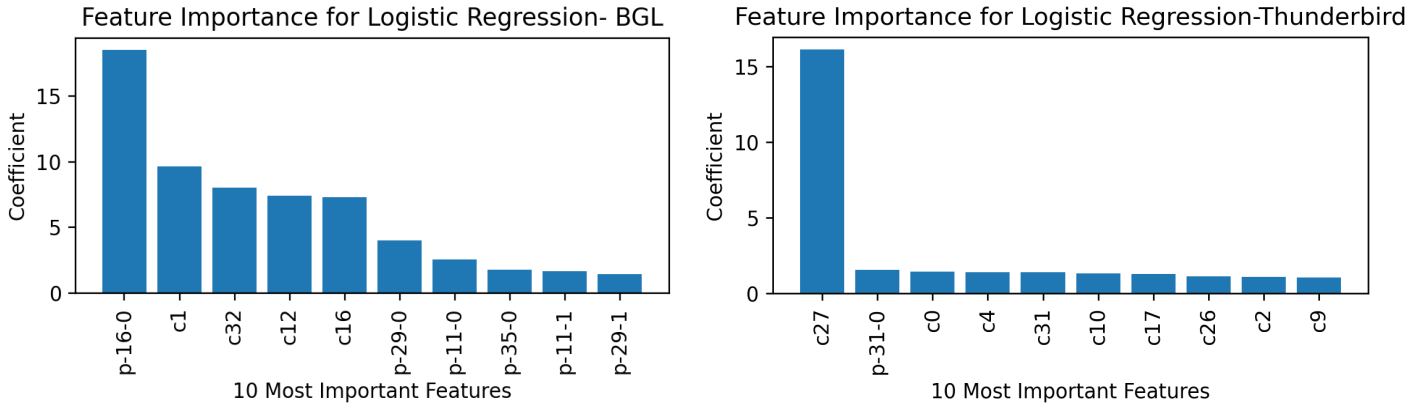

Figure 4: Features with the most predictive power for the Logistic Regression model

For BGL, all features after c16 have very low feature importance (Figure 4, left). And for Thunderbird, only cluster 27 has a significant correlation with the label (Figure 4, right). Unlike support vector machines, logistic regression does not try to maximize the margin between the decision boundary and data points on either side. As long as the linear decision boundary minimizes error, the exact value of many coefficients is not important. We see that the clusters with the highest correlation to the label also have the highest coefficients (except for p-16-0). But cluster 34 is not one of these (Figure 4, left), which is likely why precision and recall are less compared to training on TFIDF vectors.

Gradient Boosted Decision Trees (GBDT)

A GBDT model was trained for BGL and Thunderbird datasets and it performed well for both the datasets on clusters with parameters, similar to logistic regression. Table 3 below summarizes the results of this model for the two datasets.

	BGL(train)	**BGL(test)**	**Thunderbird(train)**	**Thunderbird(test)**
Precision	.9932	.8671	1.000	1.000
Recall	.9988	.9954	.9993	.9964

Table 7: GBDT performance for BGL and Thunderbird datasets (clusters with params)

The results of the GBDT algorithm are very promising, but not as high as logistic regression on BGL (Tables 3, 4). The GBDT model is able to make precise predictions on both the training datasets for anomaly detection.

The model had lower precision and recall on the test dataset than on the training dataset for BGL indicating that the model may be overfitting the training data (Table 7).

Similar to logistic regression, feature importance analysis was performed using GBDT for BGL and Thunderbird datasets. The results, shown in Figure 5, indicate that GBDT found all the clusters that were highly correlated with anomalies.

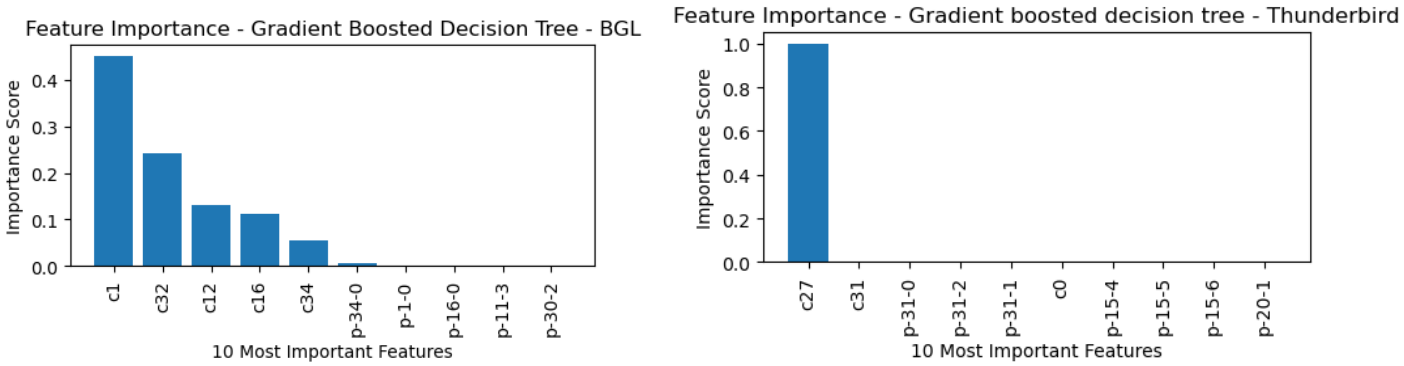

Figure 5: Features with the most predictive power for GBDT

In contrast to logistic regression, feature importances were near zero for features with little correlation to anomalies (Table 5). Rather than optimizing a linear function, decision trees select the best feature for each branch and ignore features that do not help distinguish anomalies from normal log lines.

XGBoost

For the XGBoost algorithm, we specified the initial parameters to include a tree-based model (instead of a linear model), a low max_depth to avoid overfitting, and the default learning rate of 0.3.

The model performed well on Thunderbird, but struggled on BGL. Table 8 summarizes the results. As with the logistic regression model, we performed a feature importance analysis to try and understand what was driving the model's decisions. Figure 6 displays the results of the analysis. Unlike logistic regression, XGBoost only relied on a couple of features. This is why precision was so high but recall was low. XGBoost correctly selected some clusters that are highly correlated with anomalies, so its anomaly predictions were accurate. But the model failed to select other clusters that also correlated with anomalies, causing recall to be low.

	BGL(train)	**BGL(test)**	**Thunderbird(train)**	**Thunderbird(test)**
Precision	.9899	.6628	1.000	1.000
Recall	.7354	.3011	.9997	.9964

Table 8: XGBoost performance for BGL and Thunderbird datasets (clusters with params)

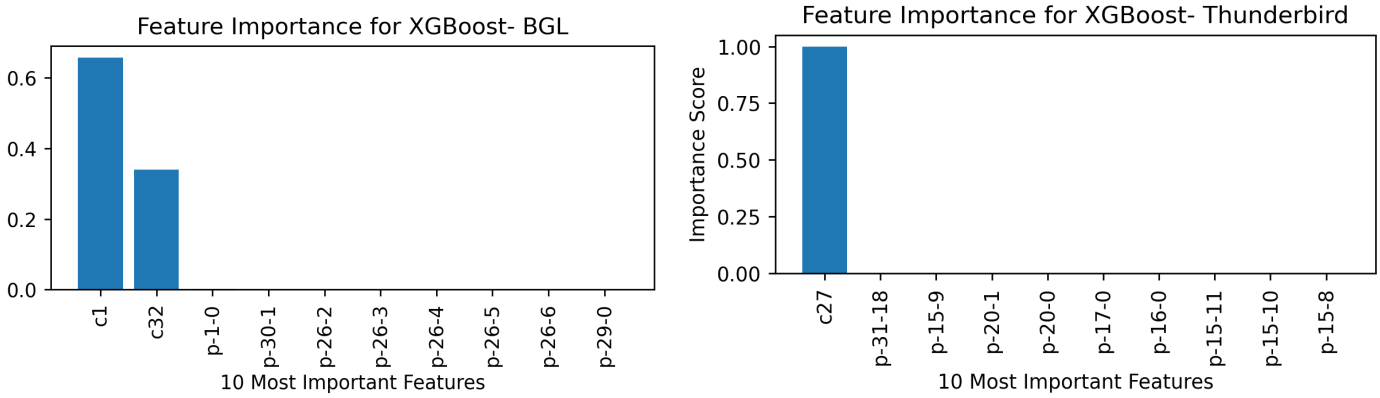

Figure 6: Features with the most predictive power for XGBoost

Unsupervised Models

Training against labels is not possible for most log sources, as creating labels is not practical for most application logs. So we also used unsupervised learning to predict the anomalous lines from TFIDF vectors. Unsupervised models don't train against labels and so typically do not make predictions. But isolation forest and one-class SVM are designed to detect outliers or anomalies, and so it is straightforward to compare their output to supervised predictions. But unlike the supervised models, unsupervised models struggled to predict anomalies in BGL and Thunderbird (Tables 9, 10).

Isolation forest and one-class SVM were trained separately with different formats of the log dataset which included TF-IDF vectorized text, log clusters (with and without parameters) and sliding window logs. Table 9 and Table 10 below summarize the results of this model and how it compares to various formats of the BGL and Thunderbird datasets.

	TFIDF raw text	**TFIDF filtered text**	**Clusters Only**	**Clusters w/Params**	**Sliding Window**
Isolation Forest	.0000	.0000	.0000	.0000	.0000
One Class SVM	.0539	.1102	.2218	.2218	.2299

Table 9: BGL Test F1 Scores by model and dataset

	TFIDF raw text	**TFIDF filtered text**	**Clusters Only**	**Clusters w/Params**	**Sliding Window**
Isolation Forest	.0000	.0000	.0000	.0000	.2512
One Class SVM	.0003	.0083	.0489	.0489	.3984

Table 10: Thunderbird Test F1 Scores by model and dataset

Isolation Forest

When the *contamination* hyperparameter was left at "auto", the isolation forest model predicted no anomalies on all datasets except the sliding window on Thunderbird (Tables 9, 10). This resulted in a precision and recall

of zero. Increasing the *contamination* shows a tradeoff between recall and precision, with the F1 score never increasing above 0.4 for BGL, and 0.5 for Thunderbird (Figure 7).

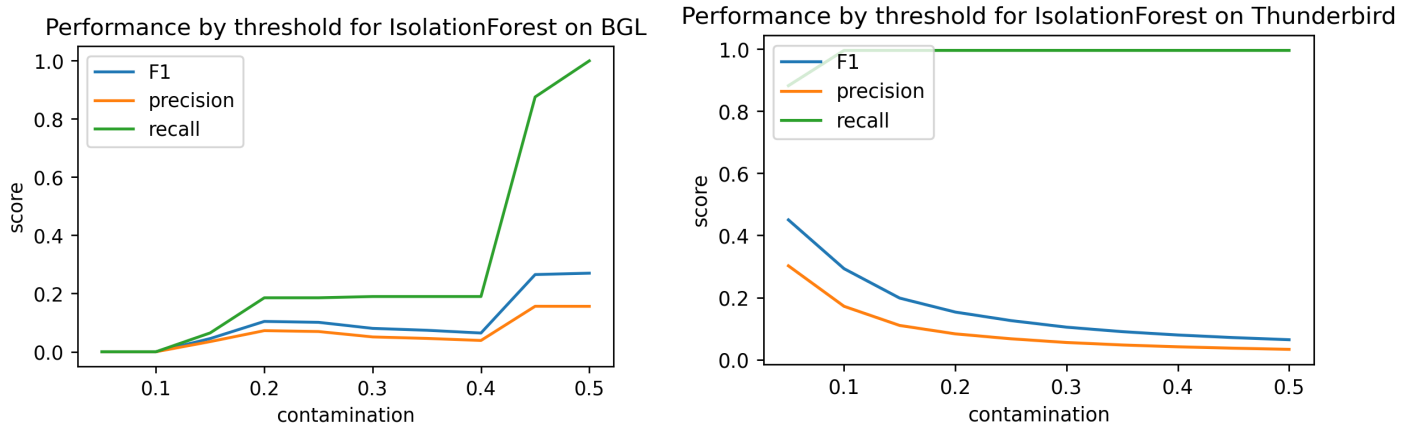

Figure 7: Performance by threshold for IsolationForest on BGL and Thunderbird

Isolation Forest struggled due to masking (Mishra, 2021). When the number of anomalies is high and forms its own cluster, Isolation Forest will not be able to detect the entire cluster as anomalous. Data points within a cluster require more partitions to be separated from each other, increasing their depth in the isolation forest, and so decreasing their probability of being labeled as anomalies. Subsampling can reduce masking by preventing anomalies from forming clusters. But the size of the anomaly clusters in BGL and Thunderbird were higher than many normal clusters. So subsampling enough to increase recall would also lower precision, since many normal clusters would be detected as false positives.

One-Class SVM

The poor F1 scores indicate that the SGD one-class SVM model did not generalize well for this dataset used for anomaly detection (Tables 9, 10). In a few scenarios, high recall with low precision indicates that a lot of normal data points (non-anomalous) are being flagged as outliers leading to the poor performance of the model. The threshold between inlier and outlier can be tuned with the *nu* hyperparameter, and this shows the tradeoff between precision and recall (Figure 8).

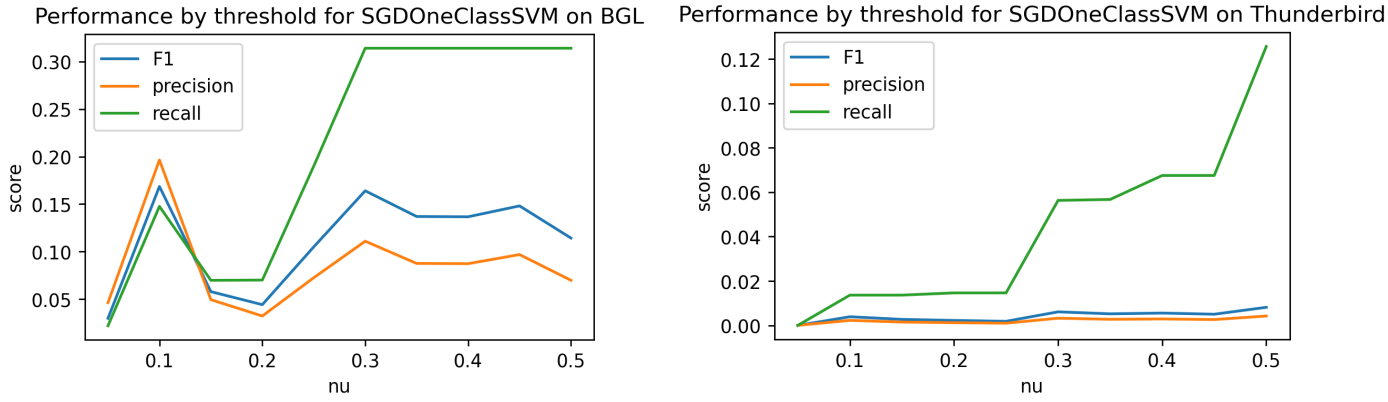

Figure 8: Performance by threshold for SGD One-Class SVM on BGL and Thunderbird

Similar to isolation forest, one-class SVM similarly struggles with masking, as large clusters of data points are likely to be within the hypersphere marking the decision boundary. A fundamental mismatch between these anomaly detection models and our datasets is that the percentage of anomalies are relatively high (see Table

1) and clustered together. One-class SVM and isolation forest are designed to find outliers, and a large cluster of related log lines is not an outlier. These anomaly detection models will not be able to distinguish the anomaly clusters from normal clusters, leading to poor model performance.

K-means

K-means groups logs into clusters. But there are many clusters, and only two labels: normal and anomaly. Compared to the ground truth labels, homogeneity of the K-means clusters is extremely high at 99.3% for BGL. That is, each cluster is mostly either all anomalies or all normal logs (Figures 9, 10).The high homogeneity means that knowing the cluster is enough to know whether a log line is anomalous. If we assign predictions based on the most likely label for each cluster, then the F1 score is 0.9973 for BGL.

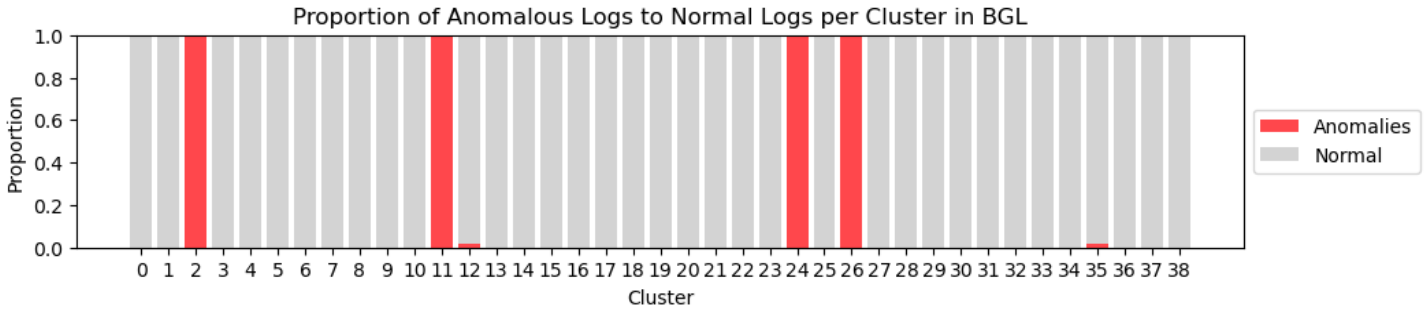

Figure 9: Proportion of Anomalous logs to Normal logs per cluster in BGL

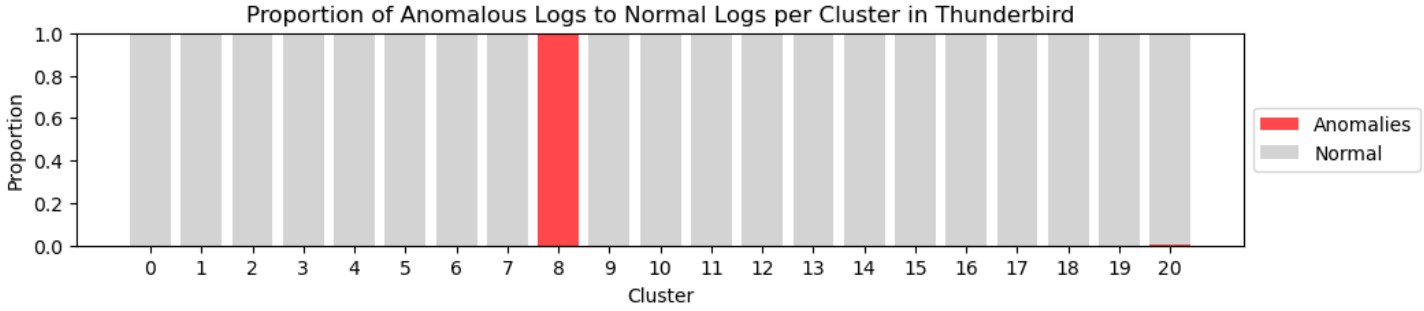

Figure 10: Proportion of Anomalous logs to Normal logs per cluster in Thunderbird

TSNE plots visually demonstrate the clear separation between most anomaly clusters and normal clusters with no false positives and few false negatives (Figures 11, 12). BGL had 42 clusters and Thunderbird had 21.

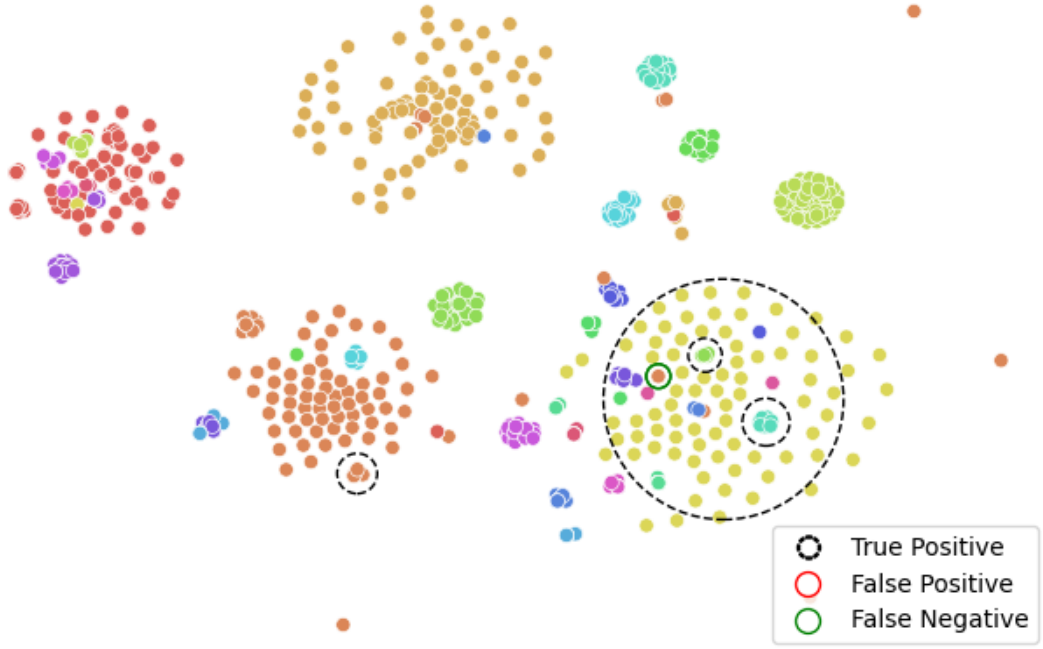

Figure 11: BGL TSNE Plot of TFIDF Vectors Colored by Cluster (perplexity = 35)

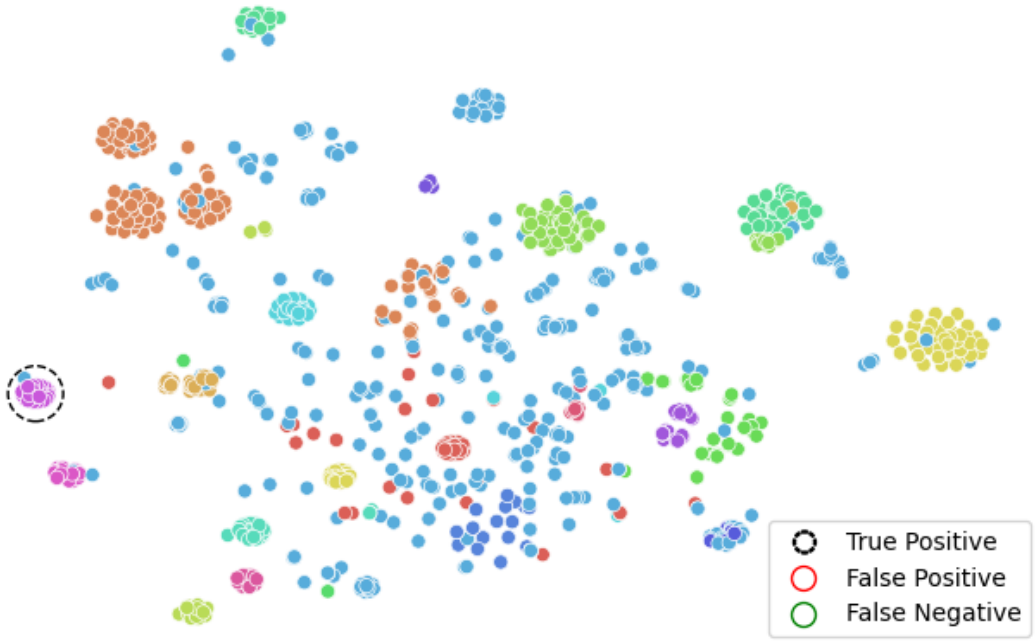

Figure 12: Thunderbird TSNE Plot of TFIDF Vectors Colored by Cluster (perplexity = 35)

Even though isolation forest and one-class SVM could not predict labels well, the predictions of one-class SVM had high homogeneity of .9654 with K-means. (Isolation forest had a perfect homogeneity of 1.000, but this was because it predicted no anomalies.) This shows that unsupervised models can tell the difference between clusters, but mostly choose the wrong clusters as anomalous. Figuring out which clusters are anomalous is very difficult, without having the labels available. We tried several criteria to distinguish normal and anomalous clusters. The first was the cluster size. But anomalous clusters are not generally the smallest clusters.

Predictability of Clusters

Next we tried predictability. Given the log line clusters right before and right after the current line, how well can a simple classifier (in our case, naive bayes) predict the current line? One hypothesis is that anomalous log lines occur randomly and so are more difficult to predict. But this hypothesis doesn't hold on BGL or Thunderbird datasets: many anomalous clusters occur hundreds or even thousands of times in a row. So it is easy to predict these anomalies based on the same anomalous line type occurring just before or after the current one. In particular, rows of the largest anomaly cluster in BGL were predictable 99.72% of the time based on the cluster assignments of surrounding rows (Table 11). We tried removing consecutive repetition to avoid this issue, but this made the most common log lines unpredictable.

Some anomalous clusters[1] (clusters 14 and 22 in Table 11) had low predictability. But many normal clusters (clusters 3 and 21 below) were even less predictable. So the predictability score tells us little about whether a cluster is anomalous.

Cluster	Anomalous?	Predictability
2	Yes	99.72%
14	Yes	3.37%
17	Yes	71.07%
22	Yes	1.85%
3	No	1.13%
21	No	0.67%
36	No	99.00%

Table 11: BGL Predictability of Selected Clusters

Another issue with predictability is that in BGL, a fatal machine check interrupt would trigger a predefined block of debug information. The log lines at the begin and end of the block were labeled as anomalous, while the rest were not. The first log line of this block ("machine check interrupt") is predictable based on what follows, and the last line ("rts panic!") is predictable based on the previous few log lines. To avoid this, predictions could be based only on previous log lines. But this would still make the last log line of the block predictable. Perhaps each entire block should be considered as a single anomaly, but that is not how the dataset is labeled.

```
KERNMC RAS KERNEL FATAL machine check interrupt
RAS KERNEL FATAL instruction address: 0x00362618
RAS KERNEL FATAL machine check status register: 0x81000000
RAS KERNEL FATAL summary.........................1
RAS KERNEL FATAL instruction plb error.............0
...
RAS KERNEL FATAL 24:1fefff40 25:0000000a 26:1fefff60 27:009741e0
RAS KERNEL FATAL 28:30001460 29:30001510 30:00000000 31:30001400
RAS KERNEL FATAL special purpose registers:
RAS KERNEL FATAL lr:003625f0 cr:20000000 xer:00000002 ctr:0037f084
KERNRTSP RAS KERNEL FATAL rts panic! - stopping execution
```

Another unpredictable log line type was correctable memory errors: "1 ddr error(s) detected and corrected on rank 0, symbol 2 over 10029 seconds". Because this occurs randomly based on hardware issues, there is no general way to predict it based on previous or following log lines. But it is not an anomaly.

[1] Note these cluster numbers used for unsupervised training differ from those used for unsupervised training. To prevent data leakage in supervised training, the clusters had to be fit only on the training set, and then predicted on the test set. Unsupervised training does not have a train/test split, and clusters were formed on the entire dataset. The clusters are similar, but labeling is different.

Cluster Distribution

Cluster	Anomalous?	Kurtosis	Skew
2	Yes	806.82	-25.45
10	No	-2.00	0.01
11	Yes	-0.72	-0.87
25	No	502.36	21.71
28	Yes	-1.99	0.00
30	No	474.43	21.31

Table 12: Statistics for Selected BGL Clusters

The third idea was to distinguish anomalous clusters based on their distribution throughout the log file. For example, non-anomalous hardware failures would be expected to occur at a predictable rate, given a large enough window. We would expect a uniform distribution of these log lines across the entire log file. We would expect anomalies to have distributions with high kurtosis, as they should be concentrated around problematic events (Figures 13, 14). We found that anomalies sometimes do have a distribution with high kurtosis, but many anomalous clusters do not have distributions with high kurtosis (Table 12). And many times the log cluster with the highest kurtosis was not anomalous. For example, in BGL, the largest anomaly cluster had the highest kurtosis (Table 12). But another anomaly cluster had the second lowest kurtosis. The other two anomaly clusters were ranked 10th and 22nd out of 40. Other statistical measures, such as skew, were also unhelpful to select anomalous clusters (Table 12). So the statistical distribution of a cluster throughout the log file is not helpful in finding anomalous clusters.

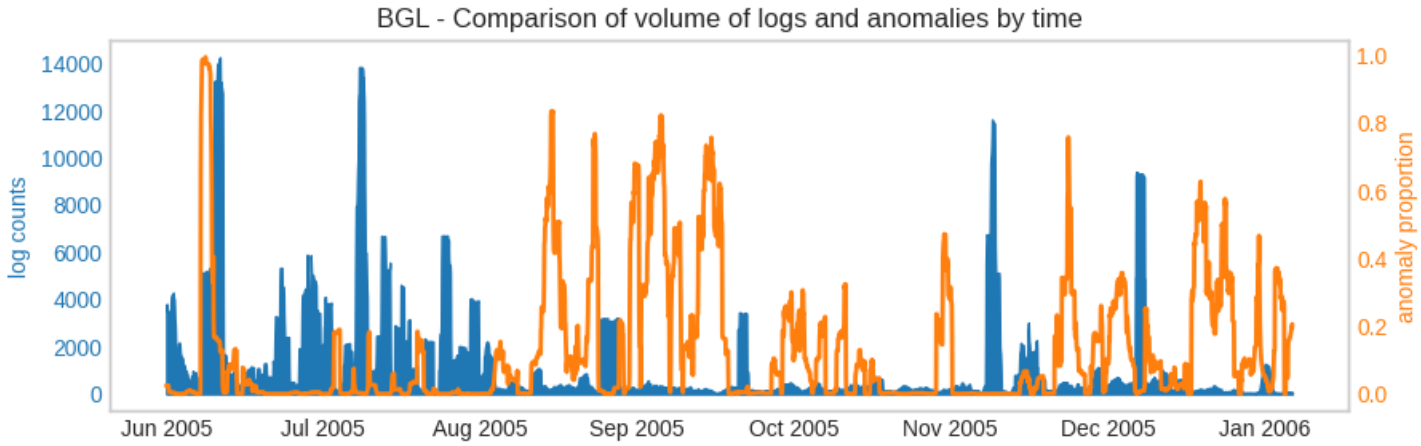

Figure 13: Comparison of volume of logs and anomalies by time in BGL

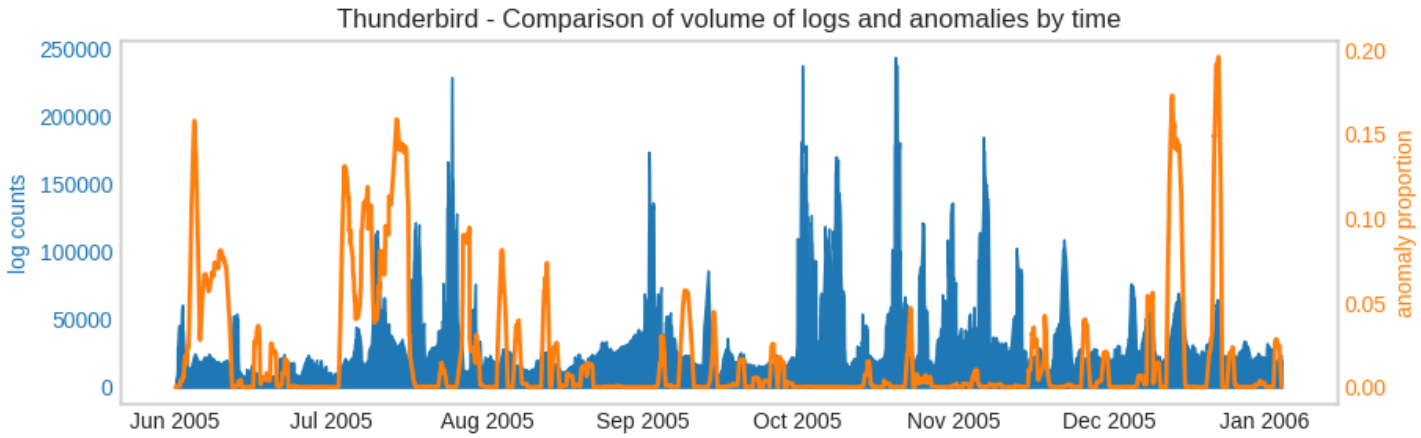

Figure 14: Comparison of volume of logs and anomalies by time in Thunderbird

We can't train against the labels because creating labels for large log files is impractical. But BGL has 4 anomalous clusters, and Thunderbird has 1 (Figures 9, 10). It is much easier to manually select a few anomalous log lines, and use these to select which clusters are anomalous. To demonstrate, we took a few anomalous log lines, sent each through TFIDF vectorization and K-means predict to find the anomalous clusters. This works as well as selecting the clusters given the label, giving a 0.9973 F1 score with minimal manual work.

K-means Mispredictions

For K-means, the F1 score was not perfect, even when assigning clusters to labels using ground truth. This is because some log statements were assigned to the wrong cluster. The reason this happens is because of the text filtering for very infrequent log lines. Here are some examples from BGL:

Original	**Filtered**
RAS APP FATAL external input interrupt (unit=0x02 bit=0x00): uncorrectable torus error	RAS APP FATAL (= =):
RAS KERNEL FATAL machine check interrupt (bit=0x06): L3 major internal error	RAS KERNEL FATAL (=): L3
RAS APP FATAL ciod: Error creating node map from file ./map-xyzt-8x8x16x1-2-mid: No child processes	RAS APP FATAL ciod: Error /-xyzt-8x8x16x1-2-mid: No

Table 13: False Negatives in BGL

Since tokens such as "interrupt" are infrequent within the cluster, they were filtered out as parameters. Then the filtered log line was used for clustering, but only had three word tokens "RAS APP FATAL". So the log line was placed into a cluster that had mostly non-anomalous logs, such as "RAS KERNEL FATAL data address: 0x4bfe7328".

Broader Impacts

With better anomaly detection from log files, it should be easier for software companies to quickly respond to unexpected events. It should also be easier for software companies to analyze large volumes of log data to find bugs and needed enhancements. Better response times and analysis can help lead to greater reliability of complex software systems, benefiting both software companies and users that depend on them.

Ethical Concerns

Logged user activity can be “anomalous” but still acceptable. With automated anomaly detection, users with different access patterns may be flagged even if they are doing nothing wrong. If the anomaly detection model is used to create adverse user events, such as locking accounts for suspicious activity, then users may be penalized for being different from the majority. Then if “anomalous” user behaviors have high correlation to protected classes, such as gender or race, the anomaly detection model could be inadvertently discriminatory.

Conclusion

For anomaly detection in BGL and Thunderbird datasets, context is not important. The best performance was achieved by considering each log line without surrounding context. Very high precision and recall was achieved with simple supervised models training TFIDF vectors against the label.

K-means clustering also works well on log files, especially with a second pass after filtering outlier tokens within each cluster. Most clusters are dense and well separated as shown by TSNE plots. The clusters have high homogeneity with the anomaly labels. Because of this these clusters can achieve high precision and recall, similar to supervised models, if it is known which clusters are anomalous. Labeling the clusters properly could be done with just a few sample anomalous lines provided by users.

But without any knowledge of which log lines or clusters are anomalous, anomaly detection is very difficult. Unsupervised models such as isolation forest and one-class SVM struggle to detect anomalous clusters, because any large cluster is seen as an inlier. For K-means, cluster properties such as size, predictability, or distribution all fail to distinguish between normal and anomalous clusters.

Future Work

Given the limitations of our current unsupervised learning models, it could prove valuable to explore alternative unsupervised learning techniques in future endeavors. Principal Component Analysis (PCA) is a good option to explore, as it can be used to construct two subspaces, one normal (constructed by k first principal components) and one anomalous (constructed by subtracting the original dimension from k) (Xu et al., 2009). Though our supervised models have performed well, it would be interesting to compare our results with those that use deep learning methods such as LSTM where the input is the characters of each log line,and the output the probability distribution of the next character which can be used to infer the abnormal possibility of the log line (Zhao et al., 2021).

Statement of Work

Manel Mahroug	● *Supervised methods*: Logistic Regression and XGBoost ● *Unsupervised methods*: Random Forest ● *Visualizations*: homogeneity bar charts ● *Report*: Abstract, parts of modeling, Literature Review
Sree Valli Maganti	● *Supervised methods*: Gradient Boosted Decision Tree ● *Unsupervised methods*: SGD One-class SVM ● *Visualizations*: Comparison of volume of logs and anomalies by

	time • *Report*: Background, Motivation, Data source, parts of modeling.
Thanuja P Stewart	• *Preprocessing:* preparsing, sampling, K-means clustering, parameter extraction and filtering, and sliding window. • *Unsupervised Learning:* K-means, cluster predictability and distribution. • *Report:* Wrote Methodology, Evaluation Strategy, Broader Impacts, and Conclusion. Contributed to Modeling, as well as Results and Analysis.

References

2.7. Novelty and Outlier Detection. (n.d.). Scikit-learn. https://scikit-learn.org/stable/modules/outlier_detection.html#overview-of-outlier-detection-methods

Akidau, T., Bradshaw, R. W., Chambers, C., Chernyak, S., Fernandez-Moctezuma, R. J., Lax, R., McVeety, S., Mills, D., Perry, F., Schmidt, E., & Whittle, S. (2015). The dataflow model. *Proceedings of the VLDB Endowment*, *8*(12), 1792–1803. https://doi.org/10.14778/2824032.2824076

Andrews, Jerone TA, Edward J. Morton, and Lewis D. Griffin. "Detecting anomalous data using auto-encoders." International Journal of Machine Learning and Computing 6, no. 1 (2016): 21.

Chen, T., & Guestrin, C. (2016). XGBoost. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. https://doi.org/10.1145/2939672.2939785

Dmlc. (n.d.). xgboost/demo at master · dmlc/xgboost. GitHub. https://github.com/dmlc/xgboost/tree/master/demo#machine-learning-challenge-winning-solutions

He, P., Zhu, J., He, S., Li, J., & Lyu, M. R. (2016). An Evaluation Study on Log Parsing and Its Use in Log Mining. *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. https://doi.org/10.1109/dsn.2016.66

Landauer, M., Onder, S., Skopik, F., & Wurzenberger, M. (2023). Deep learning for anomaly detection in log data: A survey. *Machine Learning With Applications*, *12*, 100470. https://doi.org/10.1016/j.mlwa.2023.100470

Mishra, A. (2021, December 10). Swamping and Masking in Anomaly detection: How Subsampling in Isolation Forests helps mitigate this? *Medium*. https://medium.com/walmartglobaltech/swamping-and-masking-in-anomaly-detection-how-subsampling-in-isolation-forests-helps-mitigate-bb192a8f8dd5

Thanda, A. (2023, May 11). What is Logistic Regression? A Beginner's Guide [2023]. CareerFoundry. https://careerfoundry.com/en/blog/data-analytics/what-is-logistic-regression/

Vaarandi, R. (2004). A data clustering algorithm for mining patterns from event logs. *Roceedings of the 3rd IEEE Workshop on IP Operations & Management (IPOM 2003) (IEEE Cat. No.03EX764), Kansas City, MO, USA,*. https://doi.org/10.1109/ipom.2003.1251233

Xu, W., Huang, L., Fox, A., Patterson, D. A., & Jordan, M. I. (2009). Detecting large-scale system problems by mining console logs. *Conference: Proceedings of the 27th International Conference on Machine Learning (ICML-10)*. https://doi.org/10.1145/1629575.1629587

Zhao, Z., Xu, C., & Li, B. (2021). A LSTM-Based anomaly Detection model for log analysis. *Journal of Signal Processing Systems*, *93*(7), 745–751. https://doi.org/10.1007/s11265-021-01644-4

Zhang, Y., & Sivasubramaniam, A. (2008). Failure prediction in IBM BlueGene/L event logs. *Proceedings*. https://doi.org/10.1109/ipdps.2008.4536397