A project report on

Anomaly Logs Detection

Submitted in partial fulfillment for the award of the degree of

Bachelor of Technology in Computer Science and Engineering with specialization in AI and ROBOTICS

by

RISHI PATRI (21BPS1396)

D L K Trinadh (21BAI1579)

N Sairam Gopal (21BRS1459)

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

November,2024

Anomaly Logs Detection

Submitted in partial fulfillment for the award of the degree of

Bachelor of Technology in Computer Science and Engineering with specialization in Artificial Intelligence and Robotics

by

RISHI PATRI (21BPS1396)

D L K Trinadh (21BAI1579)

N Sairam Gopal (21BRS1459)

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

November,2024

VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)
CHENNAI

DECLARATION

I hereby declare that the thesis entitled **"ANOMALY LOGS DETECTION"** submitted NIDUMOLU SAIRAM GOPAL **(21BRS1459)**, for the award of the degree of Bachelor of Technology in Computer Science and Engineering, Vellore Institute of Technology, Chennai is a record of bonafide work carried out by me under the supervision of **Dr. O S JANNATH NISHA.**

I further declare that the work reported in this thesis has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place: Chennai

Date: **Signature of the Candidate**

VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)
CHENNAI

School of Computer Science and Engineering

CERTIFICATE

This is to certify that the report entitled **"Anomaly Logs Detection"** is prepared and submitted by **NIDUMOLU SAIRAM GOPAL** (**21BRS1459**) to Vellore Institute of Technology, Chennai, in partial fulfillment of the requirement for the award of the degree of **Bachelor of Technology in Computer Science and Engineering with specialization in Artificial Intelligence and ROBOTICS** is a bonafide record carried out under my guidance. The project fulfills the requirements as per the regulations of this University and in my opinion meets the necessary standards for submission. The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma, and the same is certified.

Signature of the Guide:
Name:
Dr. JANNATH NISHA
Date:

Signature of the Examiner
Name:
Date:

Signature of the Examiner
Name:
Date:

Approved by the Head of Department,
(CSE with specialization in Artificial Intelligence and ROBOTICS)

Name: **Dr. S. HARINI**
Date:

(Seal of SCHOOL YOU BELONG TO)

ABSTRACT

The fundamental concept of log analysis for the detection of anomalies remains central to ensuring reliability, security, and operational efficiency in modern, complex software systems. These systems grow in size and complexity and produce log data on a massive scale such that it becomes impossible to manually or systematically identify any abnormal pattern that could be an indication of system malfunction, security breach, or degradation in performance. Such anomalies can be typically discovered through standard rule-based approaches; however, the methods usually struggle with large volumes, greater diversity, and increased complexity in log data. Advanced machine learning, deep learning, and NLP techniques are tools employed by researchers and practitioners to automatically and enhance the detection process.

This paper presents a wide-ranging analysis of the techniques for detecting anomalies from log files: unsupervised, supervised, and hybrid. Techniques like clustering and autoencoders are better suited if labeled data occur in greatly scarce or nonexistent amounts, for the purpose of detecting new or unexpected anomalies. Supervised Learning approaches depend on a dataset already having been labeled so it may be used as a training set for models such as decision trees, random forests, and deep neural networks to classify whether log entries qualify as normal or anomalous. In this context, hybrid approaches try to balance better between precision and flexibility by combining the interpretability of rule-based systems with the adaptability and predictive power of machine learning models.

Important challenges plaguing anomaly log detection are noisy and unstructured log data, scalability to large datasets, false positives, and real-time detection capabilities. We experiment and test the performance of many of the state-of-the-art approaches on different real-world log datasets and describe their respective strengths, weaknesses, and practical implications for real-world deployments.

Finally, we outline some possible future directions for the field, focusing on requirements for greater interpretability, better integration with the cyber security domain, and real-time processing of system logs. Evolution of software systems involves sophisticated anomaly detection methods that identify and address issues at the earliest step of the process in order to avoid potential failures.

ACKNOWLEDGEMENT

It is my pleasure to express with deep sense of gratitude to **Dr. Jannath Nisha O S**, Assistant Professor, School of Computer Science and Engineering, Vellore Institute of Technology, Chennai, for her constant guidance, continual encouragement, understanding; more than all, she taught me patience in my endeavor. My association with her is not confined to academics only, but it is a great opportunity on my part of work with an intellectual and expert in the field of Deep Learning.

It is with gratitude that I would like to extend my thanks to the visionary leader Dr. G. Viswanathan our Honorable Chancellor, Mr. Sankar Viswanathan, Dr. Sekar Viswanathan, Dr. G V Selvam Vice Presidents, Dr. Sandhya Pentareddy, Executive Director, Ms. Kadhambari S. Viswanathan, Assistant Vice-President, Dr. V. S. Kanchana Bhaaskaran Vice-Chancellor, Dr. T. Thyagarajan Pro-Vice Chancellor, VIT Chennai and Dr. P. K. Manoharan, Additional Registrar for providing an exceptional working environment and inspiring all of us during the tenure of the course.

Special mention to Dr. Ganesan R, Dean, Dr. Parvathi R, Associate Dean Academics, Dr. Geetha S, Associate Dean Research, School of Computer Science and Engineering, Vellore Institute of Technology, Chennai for spending their valuable time and efforts in sharing their knowledge and for helping us in every aspect.

In jubilant state, I express ingeniously my whole-hearted thanks to **Dr. HARINI. S**, Head of the Department, B.Tech. Computer Science and Engineering with specialization in Artificial Intelligence and Robotics and the Project Coordinators for their valuable support and encouragement to take up and complete the thesis.

My sincere thanks to all the faculties and staffs at Vellore Institute of Technology, Chennai who helped me acquire the requisite knowledge. I would like to thank my parents for their support. It is indeed a pleasure to thank my friends who encouraged me to take up and complete this task.

Place: Chennai
Date:

D L K TRINADH
RISHI PATRI
N SAIRAM GOPAL

CONTENTS

CHAPTER 4

PROTOTYPE

CHAPTER 5

IMPLEMENTATION AND RESULTS

CHAPTER 6

CONCLUSION

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

t-SNE	t-Stochastic Neighbor Embedding
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
AI	Artificial Intelligence
ML	Machine Learning
DL	Deep Learning
FP	False Positive
FN	False Negative
TP	True Positive
TN	True Negative
NLP	Natural Language Processing
TF-IDF	Term Frequency-Inverse Document Frequency
APT	Advanced Persistent Threats
PCA	Principal Component Analysis
DB-SCAN	Density-Based Spatial Clustering of Applications with Noise

Chapter 1

1.1 INTRODUCTION

While reliability and security of systems become the most important factors that make a ride easy across all industries in the modern computing environment, from infrastructure as a cloud computing system to embedded systems, such platforms result in significant volumes of log data detailing events, user interactions, system performance, and possibly errors. Logs form an important source of information to monitor the state of the system, detect issues, and potentially better monitor security threats. Although this was manageable earlier, with relatively simple IT environments, sheer volumes of logs now generated far exceed human capacity to be analyzed and detect potential anomalies that may indicate system failures or breaches or operational inefficiencies.

Log anomaly detection is the identification of known normal system behavior patterns or behaviors, which are different from these normal functions. The anomalies can only point to hardware malfunctions, software bugs, cyberattacks, or misconfigurations and certainly only can be identified in real-time to avoid larger system disruptions or security incidents. The detection of such anomalies is, however, not a very straightforward process on account of various reasons like heterogeneous nature of the logs being considered, the presence of noise in the data, dynamic behavior of the modern systems, etc.

Rules-based and threshold-based anomaly detection relies on predefined sets of rules or thresholds. These techniques are usually very effective for specific, well-defined domains, but they usually fail to scale well when applied to data that are large in size and dynamic in nature. New anomalies may not be detected for obvious reasons: those anomalies have never been seen in the training data. Third, the process of handcrafting rules to control complex and changing systems is usually quite tedious and time-consuming.

A relatively new and exciting area of research is machine learning and deep learning-one that can be applied to automate and, therefore, enhance the precision of an anomaly log's detection. Techniques like clustering, autoencoders, or even deeper neural networks may apply to construction models that will learn patterns from historical data and would be able to classify items as anomalies even without previously defined rules. The system also provides more use cases of NLP to draw meaningful insights from unstructured log entries. It further enhances the capabilities of automated anomaly detection systems.

All of this notwithstanding, much remains as difficult. Managing log data in real-time does pose a significant challenge, especially with the enormous size of data coming in. Interpreting machine learning models, eliminating false positives, and fitting anomaly detection within the established system-monitoring and security infrastructures all pose challenges as subjects of discussion. With growing

dependency on complex, distributed systems, detecting and responding to anomalies in logs will become increasingly important for system reliability and resilience.

The state of the art in anomaly log detection is discussed herein, from the most traditional techniques to the very latest machine learning-based techniques. Current challenges and limitations in the area are discussed, along with directions towards further research that may improve the effectiveness and scalability of anomaly detection systems.

1.2 OVERVIEW OF THE PROJECT

System anomaly log detection is that part of system monitoring which deals with the identification of irregular behavior, or patterns differing from the normally occurring ones in log data, possibly pointing toward faults, security breaches, or operational inefficiencies. Logs refer to any type of records generated automatically by systems, applications, and devices related to events-which may involve user activities, system performance, errors, or security incidents. These logs indicate anomalies in them which help in diagnosing issues before happening, thus preventing possible failures and enhancing the overall system's resilience. Anomalies in log data can manifest in many different ways; there are three common forms: point anomalies, contextual anomalies, and collective anomalies. Point anomalies are when one event or log entry significantly deviates from the norm pattern, an unexpected spike of error messages, a system crash at an odd time. Contextual anomalies depend on the context in which the event occurs; an event could be normal under certain conditions but abnormal with some other context. For example, high memory usage during peak business hours, but similar usage during off-peak hours may be an abnormal problem. Collective anomalies are those cases where a set of log events as a group indicate abnormal behavior, although individual events might look normal when taken in isolation. For instance, login failure attempts over time could imply, when analyzed in context, that the system was under a brute-force attack or attempt at breach. These types of anomalies illustrate potential problems and reinforce just how complex anomaly detection is when it comes to logs.

1.3 CHALLENGES PRESENT IN PROJECT

One of the challenges is anomaly log detection, especially for more complex systems, characterized by high volumes of data and dynamic behaviors. This prevents the development of efficient models, which could be effective at any anomaly occurrence. Some of the key challenges associated with it include:

1. High Volume and Velocity of Log Data:

Modern systems produce enormous volumes of log data at incredibly rapid speeds, especially in big distributed spaces like cloud platforms, microservices architectures, and IoT networks. Truly processing such enormous scales in real-time or near real-time is an extremely challenging task. How complex detection accuracy will suffer to process high-throughput streams cannot be used.

2. Variety and Complexity of Log Data:

Logs are of various types, such as structured, semi-structured, and unstructured data. The heterogeneity of log types makes it especially difficult to have an anomaly detection approach that performs reasonably good for all types of logs. Besides, text-based not structured might use techniques like NLP for meaningful extraction which usually worsens the anomaly detection problem.

3. Noisy and Redundant Data:

Logs typically contain a large amount of noise or unimportant information that negates signal strength, dominating the patterns of information and making the likelihood of false positives even higher. For instance, some logs of routine normal practices or minor harmless errors do not represent any real anomaly at all. That kind of noise has to be filtered at a level where valuable data is not lost; else, any anomaly detection system will produce low precision.

4. Dynamic System Behavior:

In dynamic environments, what is qualified as normal behavior changes with time. Updating of the system with new features, users, or infrastructure permanently changes its baseline behavior, which is challenging for static models to detect anomalies correctly. The anomaly detection system needs adaptation to these changes without creating a high false alarm rate or missing real problems.

5. Scarce Labeled Data:

Supervised learning-based models require data that is labeled; that is, logs are identified as normal or anomalous. However, in reality, log data that is actually labeled is rare or unavailable at all. It is expensive and time consuming to label large datasets manually, which has resulted in inaccurate application of supervised learning methods rather than using unsupervised or semi-supervised ones.

6. High False Positive Rate:

Overgeneration of false positives-that is, events labeled as anomalous when, in fact, they are normal-is the most significant problem that occurs with anomaly detection systems. A very large false positive rate simply overwhelms system operators with too many alerts, and an "alert fatigue" effect may occur, meaning they become too desensitized to real anomalies.
Such a balance of sensitivity (captures all true anomalies) and specificity (least false positives) remains the main challenge in developing effective detection systems.

7. Requirements for Real-Time Detection:

Many of the modern systems today demand near real-time capabilities for anomaly detection so as not to let disruptions occur, minimize downtime, or even stop security breaches. Real-time performance, analysis of huge volumes of logs, preservation of detection accuracy, and minimizing false positives pose an essentially technical challenge-to design very efficient algorithms that can quickly handle streams of data.

8. Interpretability of Detection Models:

For anomaly detection, machine learning and deep learning have proven an especially useful type of model, although such models are often "black boxes" and bring little insight into why a specific event was tagged as anomalous. Interpretability is paramount for system administrators and security teams, though-they need to know why an anomaly occurred so they can take corrective action. How to build models that are accurate but also interpretable remains an open challenge.

9. Scalability:

Systems, like anomaly detection systems, also need to scale with higher user activity, added infrastructure or more components. The giant technical hurdle comes from large datasets, complex system interactions, and higher log volumes without losing performance or in detection accuracy.

10. Integration with Other Systems:

These anomaly detection solutions must meet at integration points with the existing monitoring and security framework, to be effective. This would include compatibility with log management tools, incident response workflows, and other cyber security measures. The additional layer of complexity from the deployment adds another degree of difficulty with the integration in various tools and platforms.

11. Security Context Change Pattern:

Cyberspace attackers always change their strategy to evade detection in anomaly detection that is security-oriented. While most of the anomalies, such as the contemporary sophisticated zero-day exploits and APTs lack patterns which current models have learned to capture; these tend to not be generally seen. The anomalistic patterns mutate with the modification of attacks; this necessitates that anomaly detection systems must keep changing if they are to be effective, and this tends to be hard in the long term.

1.4 PROJECT STATEMENT

This would be a project developing an advanced anomaly detection system for complex log data. It would extend conventional and rule-based methodologies into the realm of ML and DL to provide real-time abnormal pattern detection with minimal false positives in large, dynamically changing datasets of high-volume log data. This would work on data variety, noise, and sparsity of labeled data. This will result in early system faults, breaches in security, and performance issues by the system for resilience, preventing downtime, and reliability. The solution will integrate with the existing monitoring and security frameworks to provide actionable insights that enhance operational efficiency.

1.5 OBJECTIVES AND SCOPE OF PROJECT

OBJECTIVES:

This project shall be meant by developing an anomaly log detection system. This system will detect the potential faults of the systems, the security breaches, and the operational inefficiencies existing in the system. It does so by utilizing the anomalies prevailing within the log data. In brief, it shall have aims toward:

This calls for real-time detection of anomalies in the log data using machine learning and deep learning.
Sensitive with low false positives.

It can handle enormous log data coming from heterogeneous sources-structured, semi-structured, and unstructured logs. It is dynamic towards changing system behaviors-continually learns and adjusts in a moving manner without any need for humans to intervene. They can provide interpretable results that give actionable insight so that system operators and security teams can immediately respond to the possible issues.

SCOPE:

Data Collection of Logs and Preprocessing: Sources of logs are gathered; these include system logs, application logs, and security logs-these are noisy, redundant, and even in different formats and structures or unstructured formats.

Development of Anomaly Detection Model: The idea for the project would be based on the development of a hybrid approach that would constitute both machine learning as well as deep learning. The identification of point, contextual, and collective anomalies within the log data with the help of clustering methods, autoencoders, and RNNs would be the approach.

Real-Time Detection:

The system operates in real-time. Therefore, anomalies are detected upon occurrence; hence, it will enable quicker diagnosis and response to any potential threat either to the system or security.

Evaluation Metrics:

Precision, recall, F1-score, and false positive/negative rates against which the model will be evaluated will see to a great detection of anomalies with as few false alarms as possible.

Scalability and Adaptability:

The solution would be able to scale with the growing volumes of data and adapt itself with the variation in system behavior over time to continue monitoring in dynamic environments.

Adaptation to Integrated System Applications:

An anomaly detection system would be designed to fit with the existing monitoring, logging, and security frameworks within an organization. It would therefore be deployable and usable within an organization's IT structure without much hassle.

User Interface and Alerts:

it will make the system a very intuitive means to present to administrators detected anomalies for analysis, carrying out impact, and present alerts in real time for potential risks or threats.

1.6 CONTRIBUTION TO THE PROJECT

From the above aspects, this project develops a few main contributions in the anomaly log detection system in relation to sound and efficient ones. Contributions in that regard are described as follows:

1. Framework on Log Data Collection and Preprocessing:

Develop a general mechanism for data collection that can collect logs from all sorts of different sources including; system logs, application logs, and security logs.

The techniques applied on log data are preprocessing techniques hence involving cleaning and normalization procedures, treatment of missing values, removal of noise techniques transformation of the unstructured format to a structured format for effective analytics.

2. Anomaly Detection Algorithms:

Here is K-Means, DBSCAN, autoencoders, and recurrent neural networks, in the design and implementation of these models using machine learning and deep learning to pick all types of point, contextual, and collective anomalies from log data.
Let's now fine-tune these models such that this system could catch anomalies from log data that would then indicate probable problems.

3. Real-Time log analysis and anomaly detection:

It must improve the real-time processing of logs with the detection of anomalies in real-time at the time of its occurrence. It must handle high volume and velocity too and should not degrade itself in a real-world scenario while doing anomaly detection.

4. Testing and Validation:

Develop a robust testing framework that tests the anomaly detection models according to the standard benchmarks like precision, recall, F1 score, and false positive rate. Test the model with real-world log data to confirm the best practice for real anomalous detection.

5. Scalability and Adaptability:

The system needs to scale large-scale log data and must have design capabilities given growth considerations within a distributed system. Growth trend needs to support general growth of an infrastructural organization.

This is because, over time, it continues changing its system behavior and patterns in the logged data due to constant learning and model strengthening.

6. Seamless interoperability with other monitoring infrastructures:

The system architecture should provide seamless interoperability with other existing IT monitoring and security systems where alerts and reports will be sent to the administrators or automated incident response systems seamlessly.

7. Testing and validation:

Present a good testing framework that can be used for evaluating the performance of proposed models on anomaly detection through employment of usual metrics such as precision, recall, F1-score, and false positive rate.
Test models on real log data by testing them

8. Scalability and Adaptability:

This system can easily scale to incorporate vast amounts of log data from distributed sources in multiple systems. In this regard, the system can expand based on the magnitude of an organization. This is a learning and adapting system over changing system behaviors and evolving log data patterns through continuous learning in models improvement.

9. Integration with Currently Deployed Monitoring Systems:

There should be an in-place system that can fit well into the already existing IT monitoring and security systems to allow ease in sending alerts and reports to administrators or to an automated incident response system.

10. Human Interface and Alert Mechanism:

Intuitive human interface should allow administrators view and monitor detected anomalies, monitor potential root causes, and act on them.
Intrusion Detection/Prevention- The real-time alerting system should alert the proper personnel to significant anomalies that indicate possible problems in the system or security thus prompting the needful actions

11. Documentation and Reporting:

Detailed documentation about the system design, algorithms used and the process of detecting anomalies. This way a solution becomes understood, maintained, and reproduced.

Operation experience in real-world applications show that how system reliability and security can increase based on the outcome of the system and performance on anomaly detection.

12. Interoperability with Monitoring Frameworks:

It has interoperability to integrate with any of the existing IT monitoring and security systems so that it sends alerting and reporting smoothly to administrators or automated incident response systems.

13. User Interface and Alert System:

Intuitive administrative interface with the help of which the administrator will be able to view detected anomalies, identify root causes, and take necessary actions

Liveness alerting functionality to alert everyone about significant anomalies so that deep insight into the problem can be gained, problems with the system, and security concerns solved timely.

14. Documentation and Reporting:

Record the system design and algorithms through the anomaly detection procedure, making the solution intelligible, maintainable, and reproducible.

Presenting the efficiency of the system, performances in detecting anomalies, and practical benefits in developing system reliability and security. The contributions resulting from the findings will ensure that the developed anomaly log detection system is comprehensive, scalable, and efficient for improved reliability, security, and efficiency of the operation in the target system.

Chapter 2

2.1 BACKGROUND WORK OF THE PROJECT

In modern complex systems, logs are continuously generated as a byproduct of system operations and therefore capture precious information regarding activities, errors, performance metrics, and security events. Such logs are essential for system administrators, the security teams, and DevOps engineers to be abreast of the state of health and security of their systems. Due to the high volume of log data and its complexity, the need to manually detect abnormal patterns is quite difficult. The increasing level of automation in large-scale infrastructures calls for the development of smart systems that self-detect anomalies in real-time.

Log-entries with anomalous behavior are usually instances that greatly deviate from the normal functioning of the system and results are often traced to system faults, security breaches, or bottlenecks in performance. Traditional approaches applied to the detection of such anomalous behavior primarily focus on predefined rules and thresholds, which are far from ideal. Rule-based systems do not adapt; update the rule base rather than the behavior of the system, require continuous update, and are prone to false positives. Further, they could never scale to handle the volume of log data most modern systems generate.

Towards that aim, unsupervised learning algorithms form a very promising methodology for anomaly detection in log data. These algorithms do not require labeled data and are expected to learn from normal behavior embedded in the structure and patterns of the data. The machine's ability to detect subtlety, complexity, and previously unknown anomalies makes it a perfect solution for real-time monitoring of logs as well as automatic incident response.

This project combines a number of unsupervised learning methods to ensure accurate and robust anomaly detection. The first algorithm employed is called "K-Means Clustering", where similar log entries are grouped. Those log entries not fitting well into any available clusters are marked as anomalies. This is, hence, a method very effective at identifying those outliers by means of distance metrics and useful in identifying unusual log patterns.

Next, t-SNE applies another type of dimensionality reduction and data visualization. Where K-Means assists in pointing out cluster formations of normal data, t-SNE allows for visualization in two or three dimensions, hence easily see anomalies that would otherwise not be clear in higher dimensional spaces. From visual inspection of the log distribution, t-SNE gives insight into patterns associated with normal and anomalous behavior.

Isolation Forest is another highly powerful unsupervised algorithm that is used in anomaly isolation, partitioning of data. In this model, the decision trees are created, splitting the data randomly, and anomalies are found based on how isolated they are against normal instances. This has turned out to be a very efficient algorithm in high-dimensional datasets and has done well in identifying rare events in log data.

To further delineate anomaly detection, one introduces One-Class SVM. One-Class SVM learns a boundary around the normal log entries and flags anomalies as any data points lying outside of this boundary. It is rather effective if normal data are plentiful but labeled anomalous data scarce.

To increase the performance and minimize false positives, "Stacking Classifiers" are used. Stacking classifiers is an ensemble learning technique where it collects the predictions of several models (K-Means, Isolation Forest, One-Class SVM) to yield an improved final prediction. The benefits of stacking classifiers are that it enables effective type handling of anomalies by the system across different log sources.

Lastly, the project also encompasses "baseline classifiers" and "ensemble methods" in order to benchmark the performance of more complex models and assess the impact of combining results from multiple detection methods. All these enhanced robustness and scalability features allow for real-time processing of considerable volumes of various log data.

The project will develop an anomaly detection system that is accurate and scalable but also adaptive to evolving behaviors of the system-a very critical tool for proactive monitoring, fault detection, and security threat mitigation of complex IT environments.

Related Study

2.2 BACKGROUND STUDY TABLE

S. No	Title	Methodology	Takeaways
1.	Adanomaly: Adaptive Anomaly Detection for System Logs with Adversarial Learning	BiGAN Model: Uses Bidirectional GANs for feature extraction; improves detection accuracy. Ensemble Learning: It develops multiple classifiers to address class imbalance and reduce reliance on hyperparameters. Log Parsing: The Drain method is used to transform raw log data into accurate templates. Feature Extraction: Features are drawn from log sequences with BiGAN and balanced data for training classifiers.	Adanomaly Framework: Proposes a new log-based anomaly detection approach using BiGAN and ensemble learning, wherein the accuracy and class imbalance are enhanced. Feature Extraction: BiGAN provides features based on reconstruction and discriminative losses to enhance detection precision. Experimental Results: It is superior recall and accuracy
			compared to six baseline methods across three public datasets.

2.	Anomaly Detection on Servers Using Log Analysis	Log Parsing: employ Drain algorithm to produce structured raw log data. Feature Extraction: •Event Count/TF-IDF: collect counts and apply TF-IDF. •Sliding Window Counts: produces matrices from sequences of events. •Final Matrix: merges sliding window matrices with TF-IDF vectors. Anomaly Detection Model: employ the CNN model trained on labelled log data.	Deep Learning Model: CNN achieved around 99% accuracy on anomaly detection. Log Parsing: Structured the raw log data via the Drain algorithm. Feature Extraction: Extracted features as follows: TF-IDF, Sliding Window Event Counts. Experimental Results: Great performance with low error rates and high F1-scores.
3.	LogST: Log Semi-supervised Anomaly Detection Based on Sentence-BERT	Log Parsing: Converts raw logs into structured templates using the Drain method. Semantic Embedding: Uses Sentence-BERT (SBERT) for semantic representations of log events. Clustering: Applies HDBSCAN for	LogST: Semi-supervised log anomaly detection using SBERT and GRU. Improved Accuracy: Outperforms traditional methods through semantic relationships. Stability with Few Labels: Effective with limited

		clustering log sequences based on semantics. Anomaly Detection: Implements a GRU neural network with semi-supervised learning for anomaly detection.	labeled normal logs. Experimental Validation: Shows significant improvements on the HDFS dataset.
4.	Machine Learning to Detect Anomalies in Web Log Analysis	Two-Level ML Algorithm: Classification uses decision tree, whereas anomalies involve the usage of HMMs. Feature Extraction: It extracts HTTP status codes and URL length as well as counts of parameters from logs. Data Labeling: It automatically identifies log attacks versus normal behavior. Performance Assessment: Uses accuracy, precision, FPR and TPR, compared against Logistic Regression and SVM.	Anomaly Detection System: Two-level algorithm comprising of a Decision Tree and HMM for detection of web logs High Accuracy Achieved 93.54% accuracy with 4.09% false positive rate outperforming Logistic Regression and SVM. Utilized Real-World Data Tested on real industrial environment data Future Improvements Adds a retraining module that would make the adaptation of the model to new attacks. patterns.

5.	Log-based Anomaly Detection Without Log Parsing	Data Splitting: 80% training and 20% testing, with unseen log messages in the test set. Sliding Window: 20-message length with a step size of 1 for log sequences. Comparison of Methods:	NeuralLog Approach: Transforms raw logs into semantic vectors using BERT, bypassing parsing. Results: Achieves F1-scores over 0.95 on four datasets, outperforming existing
		NeuralLog compared to SVM, LR, IM, LogRobust, and Log2Vec. Evaluation Metrics: Uses Precision, Recall, and F1-score across datasets (HDFS, BGL, Thunderbird, Spirit).	methods. Contributions: Highlights limitations of log parsing and NeuralLog's effectiveness with OOV words.
6.	A Study on Log Anomaly Detection using Deep Learning Techniques	Feature Extraction: TF-IDF and Word2vec are used to transform the log data into dense vectors. Machine Learning: It utilizes SVM, Decision Tree, and PCA for the detection of anomaly. Deep Learning: Utilizes LSTM, RNN, Autoencoder, and Bi- LSTM for advanced anomaly detection.	Importance of Anomaly Detection: Vital for reliability and performance in large-scale networked systems. Deep Learning Techniques: The models used include RNN, LSTM & autoencoders to create a method of effective log anomaly detection. Challenges: Unclean data complications, log instability, and log bursts.

7.	A Comprehensive Review of Anomaly Detection in Web Logs	Rule-based Models: Detect known anomalies but rely on administrator expertise. Statistical Models: Use Regex for anomaly detection based on query parameters. Supervised & Unsupervised ML Models: Supervised methods use labeled data;	Focus on Web Logs: Reviews techniques for detecting anomalies in HTTP logs. Categorization: Classifies methods into rule-based, statistical, supervised, unsupervised, and deep hybrid models. Challenges: Discusses issues
		unsupervised methods employ clustering for unknown anomalies. Deep Hybrid Models (DHM): Combine log sequence encoding and machine learning for semantic anomaly detection.	like high dimensional data and adaptive thresholds. Applications: Highlights use in cybersecurity, including IDS and FDS.

8	Detecting Large-Scale System Problems by Mining Console Logs"	The paper presents an anomaly detection method for massive distributed systems through mining of console logs. That work is focused primarily on message-parsing and feature extraction techniques used to find patterns for failure or irregularities in the system.	Console logs are important sources of identifying problems in distributed environments. Examining these logs proves useful in early detection of problems resulting in better system reliability. Relevant features extracted from the logs make it easy to pinpoint anomalies that may eventually cause system failures and require intervention before their occurrence.
9	Spell: Streaming Parsing of System Event Logs	Du and Li suggested Spell: the first streaming approach toward real-time parsing of system event logs. The approach offers on-the-fly processing, thus making log analysis faster and more efficient in large-scale systems.	This reduces time complexity and efficiently takes care of the system event logs. The approach supports the real-time detection of an intrusion and system errors. The presence of timely intervention enhances system security and performance.
10	Log-based Predictive Maintenance for Cyber-Physical Systems Using Machine Learning	Kim, Cummins, and Abraham focus on the use of log data for predictive maintenance for cyber-physical systems, investigating anomaly detection as a precursor to early system failure prediction using machine learning models.	Log-based predictive maintenance improves operational resilience and reliability. The machine learning models provide good accuracy for anomaly detection and failure prediction. The approach helps avoid the performance impacts because it considers the resolution of problems prior to their occurrence.
11	A Deep Learning Approach for Network Log Anomaly Detection	Zhang et al apply deep learning models, in particular RNN and LSTM networks, to network log anomaly detection. This will be a more effective approach than traditional techniques because the models do take into consideration the sequential nature of data patterns.	Zhang et al. applied the deep learning method using RNN and LSTM models for network log anomaly detection as they capture sequential patterns that do better than a traditional technique that would not detect unusual activities

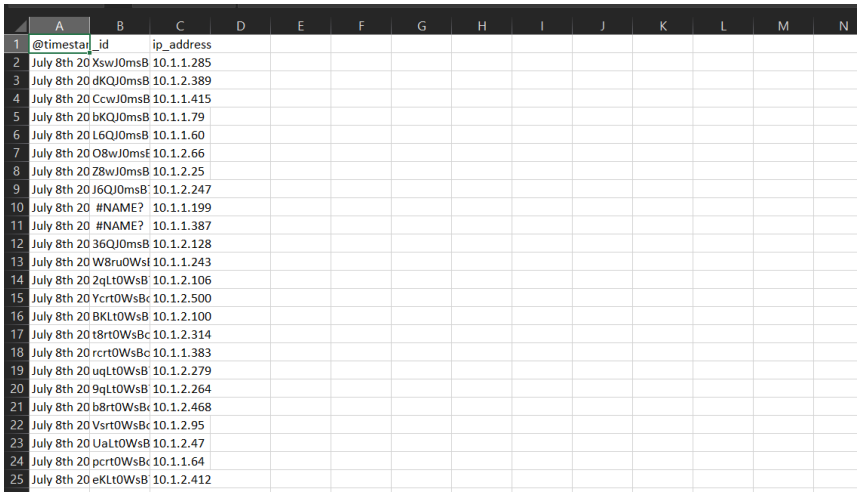
			possibly pointing towards some breach or misconfigurations in systems.
12	Log-based Anomaly Detection without Log Parsing	Zhu, He, He, and Lyu approach a different method of log-based anomaly detection that obviates traditional log parsing. Rather, the authors make use of a sequence-to-sequence model that predicts log entries as it detects anomalies based on discrepancies between predicted and actual logs.	This method improves flexibility because it gets rid of the dependence on log parsing, achieving high accuracy about anomalies through prediction errors. Direct prediction of log entries offers the approach as much more efficient and scalable solution for anomaly detection than the extensive log preprocessing over prerequisites.
13	Anomaly Detection in Smart Grid Data Using Hybrid Machine Learning Models	Kavousi-Fard and Samet attempt to merge anomaly detection in smart grid data using a combined machine learning approach. They combine different algorithms such as Isolation Forest for increasing detection accuracy, especially in large amounts of data in real-time, with clustering.	This hybrid approach improves anomaly detection accuracy in smart grid data by leveraging the strengths of multiple machine learning techniques. In addition, it follows scalability, and therefore the technique is effective for processing large amounts of real-time data. It is especially useful in network-based log anomaly detection, where handling high data volumes is crucial for timely issue identification.

Chapter 3

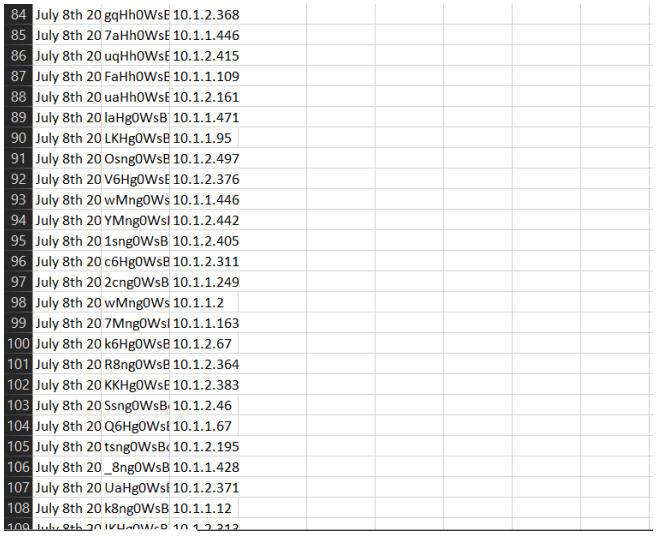
Proposed Architecture

3.1 DATASET

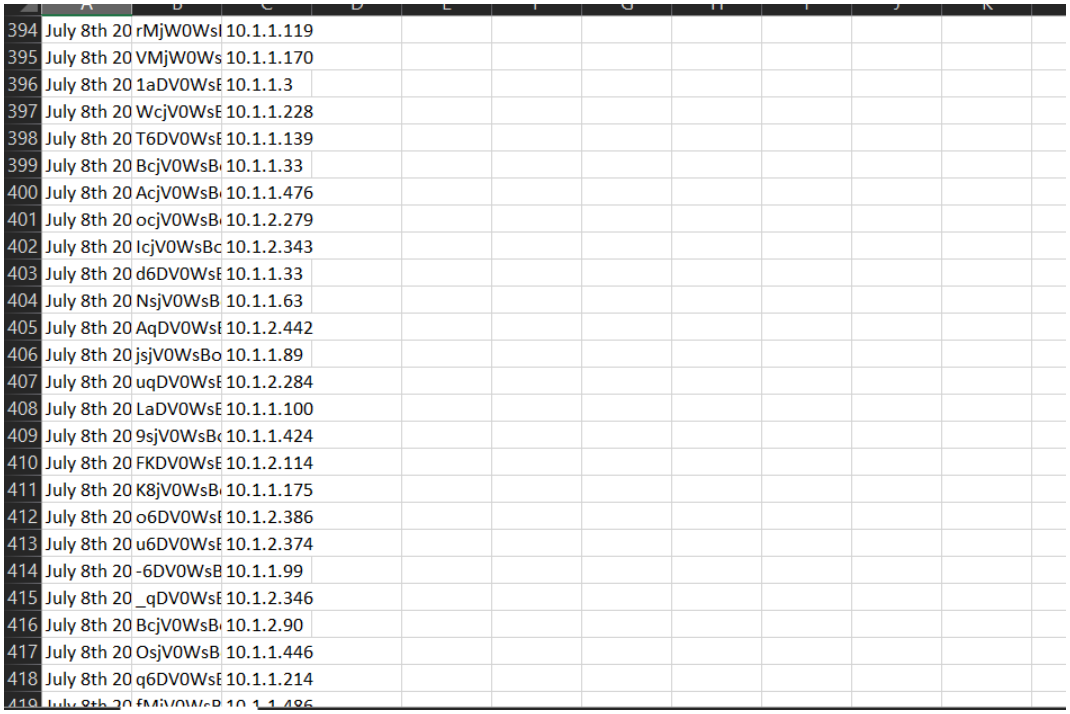
We use a log entry dataset in this project. The log entries are categorized into normal logs and anomalous logs whose categories may include system failures, security breaches, and unusual network patterns. We have divided the dataset into three sections: train, valid, and test, while putting percent to 87%, 8%, and 4% respectively for train, valid, and test. We utilized automated techniques of anomaly detection to pre-process and label the logs, and trained our detection model using Google Colab / Jupyter notebook.



	A	B	C
1	@timestar	_id	ip_address
2	July 8th 20	XswJ0msB	10.1.1.285
3	July 8th 20	dKQJ0msB	10.1.2.389
4	July 8th 20	CcwJ0msB	10.1.1.415
5	July 8th 20	bKQJ0msB	10.1.1.79
6	July 8th 20	L6QJ0msB	10.1.1.60
7	July 8th 20	O8wJ0msB	10.1.2.66
8	July 8th 20	Z8wJ0msB	10.1.2.25
9	July 8th 20	J6QJ0msB	10.1.2.247
10	July 8th 20	#NAME?	10.1.1.199
11	July 8th 20	#NAME?	10.1.1.387
12	July 8th 20	36QJ0msB	10.1.2.128
13	July 8th 20	W8ru0WsB	10.1.1.243
14	July 8th 20	2qLt0WsB	10.1.2.106
15	July 8th 20	Ycrt0WsB	10.1.2.500
16	July 8th 20	BKLt0WsB	10.1.2.100
17	July 8th 20	t8rt0WsB	10.1.2.314
18	July 8th 20	rcrt0WsB	10.1.1.383
19	July 8th 20	uqLt0WsB	10.1.2.279
20	July 8th 20	9qLt0WsB	10.1.2.264
21	July 8th 20	b8rt0WsB	10.1.2.468
22	July 8th 20	Vsrt0WsB	10.1.2.95
23	July 8th 20	UaLt0WsB	10.1.2.47
24	July 8th 20	pcrt0WsB	10.1.1.64
25	July 8th 20	eKLt0WsB	10.1.2.412



84	July 8th 20	gqHh0WsB	10.1.2.368
85	July 8th 20	7aHh0WsB	10.1.1.446
86	July 8th 20	uqHh0WsB	10.1.2.415
87	July 8th 20	FaHh0WsB	10.1.1.109
88	July 8th 20	uaHh0WsB	10.1.2.161
89	July 8th 20	laHg0WsB	10.1.1.471
90	July 8th 20	LKHg0WsB	10.1.1.95
91	July 8th 20	Osng0WsB	10.1.2.497
92	July 8th 20	V6Hg0WsB	10.1.2.376
93	July 8th 20	wMng0WsB	10.1.1.446
94	July 8th 20	YMng0WsB	10.1.2.442
95	July 8th 20	1sng0WsB	10.1.2.405
96	July 8th 20	c6Hg0WsB	10.1.2.311
97	July 8th 20	2cng0WsB	10.1.1.249
98	July 8th 20	wMng0WsB	10.1.1.2
99	July 8th 20	7Mng0WsB	10.1.1.163
100	July 8th 20	k6Hg0WsB	10.1.2.67
101	July 8th 20	R8ng0WsB	10.1.2.364
102	July 8th 20	KKHg0WsB	10.1.2.383
103	July 8th 20	Ssng0WsB	10.1.2.46
104	July 8th 20	Q6Hg0WsB	10.1.1.67
105	July 8th 20	tsng0WsB	10.1.2.195
106	July 8th 20	_8ng0WsB	10.1.1.428
107	July 8th 20	UaHg0WsB	10.1.2.371
108	July 8th 20	k8ng0WsB	10.1.1.12



394	July 8th 20	rMjW0WsI	10.1.1.119
395	July 8th 20	VMjW0Ws	10.1.1.170
396	July 8th 20	1aDV0WsE	10.1.1.3
397	July 8th 20	WcjV0WsE	10.1.1.228
398	July 8th 20	T6DV0WsE	10.1.1.139
399	July 8th 20	BcjV0WsB	10.1.1.33
400	July 8th 20	AcjV0WsB	10.1.1.476
401	July 8th 20	ocjV0WsB	10.1.2.279
402	July 8th 20	IcjV0WsBc	10.1.2.343
403	July 8th 20	d6DV0WsE	10.1.1.33
404	July 8th 20	NsjV0WsB	10.1.1.63
405	July 8th 20	AqDV0WsE	10.1.2.442
406	July 8th 20	jsjV0WsBo	10.1.1.89
407	July 8th 20	uqDV0WsE	10.1.2.284
408	July 8th 20	LaDV0WsE	10.1.1.100
409	July 8th 20	9sjV0WsB	10.1.1.424
410	July 8th 20	FKDV0WsE	10.1.2.114
411	July 8th 20	K8jV0WsB	10.1.1.175
412	July 8th 20	o6DV0WsE	10.1.2.386
413	July 8th 20	u6DV0WsE	10.1.2.374
414	July 8th 20	-6DV0WsB	10.1.1.99
415	July 8th 20	_qDV0WsE	10.1.2.346
416	July 8th 20	BcjV0WsB	10.1.2.90
417	July 8th 20	OsjV0WsB	10.1.1.446
418	July 8th 20	q6DV0WsE	10.1.1.214
419	July 8th 20	fMjV0WsB	10.1.1.486

721537	June 9th 2	SxuJOWsB	10.1.1.191
721538	June 9th 2	okWJOWs	10.1.2.398
721539	June 9th 2	1UWJOWs	10.1.2.343
721540	June 9th 2	ERuJOWsE	10.1.1.486
721541	June 9th 2	NEWJOWs	10.1.2.382
721542	June 9th 2	dxuJOWsB	10.1.2.191
721543	June 9th 2	jhuJOWsB	10.1.1.350
721544	June 9th 2	dkWJOWs	10.1.2.66
721545	June 9th 2	TBuJOWsE	10.1.1.249
721546	June 9th 2	vUWJOWs	10.1.1.200
721547	June 9th 2	aUWJOWs	10.1.2.432
721548	June 9th 2	AEWIOWs	10.1.2.156
721549			

3.2 K- Means Clustering

K-Means Clustering is one of the most commonly used unsupervised machine learning algorithms that guarantees division of the dataset into different clusters based on similarity. With K-Means being applied to anomaly log detection, it basically groups the log entries together based on their features and behavior to see patterns and flag anomalies that are well out from what is considered normal groupings.

How K-Means Clustering Works:

Steps 1:

The algorithm starts with a predefined number of clusters, then it randomly picks K initial centroids (representatives of each cluster).

2. Assignment:

Assign every data point (or log entry in your case) to the nearest centroid using some distance metric, usually Euclidean distance.

3. Update Centroids:

After the assignment step recalculate centroids by averaging all data points
Points in a cluster. The process is repeated until the centroids do not change, or the algorithm converges.

4. Convergence:

The iteration of mapping points to clusters and updating centroids continues until convergence. It is when the centroids move or change very little.

5. Anomaly Detection:

Once the clustering has been done, entries that are far from the centroids or do not fall very well into any of the clusters-they are small or outlying clusters-are flagged out as anomalies. This anomaly may be due to some error in the system, security breach, or operational errors.

Key Elements:

K (number of clusters):

It is one of the critical hyperparameters used in K-Means. Normally, the value of K can often be deduced using methods like the Elbow Method or Silhouette Score.

Distance Metric:

Euclidean distance is normally used in finding similarity between any points of data, though sometimes Manhattan distance has to be used depending upon the problem being tackled.

Centroid:

The centroid: it is the middle of the cluster or the average point assigned to that cluster.

Applications in Anomaly Detection:

Cluster Formation: K-Means groups similar log entries together. Normal log entries should produce well-defined clusters, and anomalies may maybe form their own little, isolated clusters or fall far from any cluster's centroid.
Anomaly Scoring: Logs far from centroids are more likely to be anomalies because they fail to map into general patterns seen in normal data.

Advantages:

- Easy, and direct; it is simple to implement.

- In case the number of clusters is set correctly, then excellent at finding large-scale anomalies
- It works perfectly well with huge data.

Disadvantages:

-Sensitive to the initial placement of centroids

-Number of clusters K should be predetermined.

-It may not work well with clusters of many shapes and densities.

Example in Anomaly Log Detection:

For example, if this dataset consists of the log of a server application, then K-Means will find one cluster consisting of logs of normal operation of the server and one or more different clusters in case of error, crash, or traffic spike, that may be tagged for further analysis and remediation as an anomaly.

3.3 Isolation Forest

Isolation Forests: It is an unsupervised algorithm for machine learning; therefore, very efficient for anomaly detection. In fact, it is particularly effective in working with large databases of high-dimensional features, like system logs. The Isolation Forest algorithm is based on the principle that anomalies have to be isolated rather than the normal data profiles. It does so by recursively partitioning the dataset and finding out those data points, which are relatively easily isolated.

How Isolation Forest Works:

1. Point Isolation

The algorithm initiates by picking a feature randomly and then a split value randomly within the minimum and maximum values of that feature. This forms "trees" recursively. This isolates one data point from the rest of the data with each split.

2. Tree Building:

Isolation Forest creates multiple trees, or decision trees, by recursive partitioning of the data set. The number of trees is a hyperparameter, and each tree is built separately with random features and split values.

3. Anomaly Scoring:

There is the Isolation Score which presumes that outliers are easier to isolate than normal points. In this regard, therefore, data points that require fewer splits to be isolated have a higher anomaly score.

-This scoring anomaly is achieved depending on the number of splits that should be crossed from any given point to isolate a data point. Ideally, most anomalies would generally have fewer path lengths because they are further apart from most points.

The score is then used for pointing whether a point is an anomaly:

Anomalous points would have a short path length meaning that they can easily be isolated.

Normal points would have greater path lengths, thus cannot be isolated alone.

4. Thresholding:

After obtaining isolation scores, then threshold can now be applied to differentiate normal data point and anomalies. The threshold used rests partially on the application to the tolerance for false positives or false negatives.

Isolation Forest Key Characteristics:

The algorithm is unsupervised. No labeled data is needed and hence it can identify anomalies from unlabeled data sets.

-Efficient: It would handle volumes with high-dimensional features well because the low time complexity, compared to most anomaly detection techniques, would allow it to work in an efficient manner.

Scalable: With the tree-based structure used, it scales well with large datasets and high-dimensional data, thus very ideal for log data with a lot of features.

Application in Anomaly Log Detection:

Isolation Forest is very good for anomaly log detection in terms of identifying the existence of unusual patterns or outliers in log data. Logs are always full of timestamped events, error messages, system usage statistics, or other system-generated entries. It can find these things below:

Unusual events:

These may include sudden spikes in error messages, invalid login attempts, or abnormal system behaviors.

For example, log entries such as access requests from suspicious IP addresses that are obviously much apart from what is considered normal will be flagged as anomalies by Isolation Forest.

More applicative usage in Anomaly Log Detection

An Isolation Forest can be very efficient in anomaly log detection, especially when it comes to recognizing the presence of unusual patterns or outliers in log data. Logs are time-stamped events, error messages, system usage statistics, or other entries that a system may generate. The algorithm may find and alert to the following:

Unusual events: These include unexpected errors or failed login attempts apart from abnormal systems.

Outlier behavior: For instance, if there exist some log entries-for example, access requests from some suspicious IP addresses-which stand out as outliers, then Isolation Forest marks them as anomalies.

Steps to Apply Isolation Forest on Log Data

1. Pre-processing of the Logs:

Log data normally requires cleaning and pre-processing to enable the application of any machine learning model. This includes transformation of categorical features to numerical features. For example, one-hot encoding or label encoding and feature standardizing of the numerical features.

2. Training the Isolation Forest Model

Isolation Forest learned atop of a log dataset; once the data is prepared, it learns to identify normal and anomalous behavior based on the structure of the data.

3. Anomaly Detection:

It is then used to predict which of the log entries would be anomalous after training. These are then highlighted for further observation.

4. Post-processing and Evaluation:

Further, the flagged anomalies can then be validated further by possibly integrating with other techniques like clustering or rule-based systems to establish the anomalies. Precision, recall, and F1-score can be used as the evaluation metric to check how well the model performs regarding the anomalies that occur in real life.

Advantages:

-Efficient and Fast: Especially applied for high-volume log data.

-No meaningful Labels: Although it is an unsupervised technique, one can apply it even when there are no labeled data in a given scenario.

-Manage high dimensionality: Effective even for very complex log data with many features or fields.

Drawbacks

-Makes an assumption on Rarity and Differentness of Anomalies: Anomaly detection algorithms typically assume that anomalies are rare and significantly different from normal data; this may not be the case in many.

-Highly Sensitive to Parameter Choices: The model output can be very sensitive to the number of trees chosen and the contamination rate that is the fraction of outliers in the dataset.

Application Example:

For example, think of a server log containing login attempts from a user. In a normal scenario, login entries generally are expected and therefore predictable. But if there's an abrupt jump in the failure rate of login attempts or login requests originated from a suspicious IP, Isolation Forest would actually isolate such anomalies in terms of how "easy" it is to segregate those entries from the rest of the log data.

3.4 T-SNE

One of the most popular dimensionality reduction algorithms is t-SNE. It often used for visualization at two or three dimensions if high-dimensional data are employed. Not a natively anomaly detection algorithm, t-SNE can be useful in anomaly log detection where complex log data may be visualized and understood. Anomalies may then be found because high-dimensional log features can easily be projected into lower-dimensional spaces in search of clusters and outliers.

How t-SNE Works:

1. Similarity Calculation:
T-SNE initializes by computing the similarities between pairings of data points in high-dimensional space. It makes use of probabilities instead of distances: it computes how likely it is that one point would be a neighbor of another given their distance in the original space.

2. Low-Dimensional Embedding:
Then, t-SNE searches for a lower-dimensional data representation, typically 2 or 3D, where points similar in high dimensional space stay close to each other and dissimilar points spread out. The algorithm reduces a cost function on the difference of similarities between a high dimensional space and a low-dimensional embedding.

3. Stochastic Process:
t-SNE is based on a stochastic process and the process is actually driven by random elements, so it proves to be very effective in generating visualizable maps for high dimensional data with achieving preservation of local structure, that is, the clusters of similar points.

4. Visualization:
Once the data has been reduced to two or three dimensions, it can be visualized on a scatter plot. Points that are similar to each other will form clusters, while outliers or anomalies will appear isolated from the main clusters.

Application in Anomaly Log Detection:

In the context of anomaly log detection, t-SNE can be used to analyze log data in the following ways:

1. Visualizing Log Data:
Logs are highly dimensional with hundreds of features like timestamp, user ID, error codes, log messages, etc. Dimensionality can be reduced using t-SNE to view and understand the relationship of different entries in the log. This is essentially helpful in picking out clusters of

normal logs and picking out outliers that indicate anomalous behavior.

2. Identifying Anomalies:
Once the log data is mapped to either 2D or 3D, it may become easier to detect anomalies if one looks at the entries away from the main clusters. These entries seem isolated to the majority of the data; they are flagged as possible anomalies. Examples of these outliers are such items as foreign login attempts without the capability to sign in and system errors that cannot be expected at the time of their appearance.

3. Cluster Analysis:
Using t-SNE to dimensionality reduce, one could, therefore have an easier way to apply direct clustering algorithms like K-Means or DBSCAN on the visualized data. Grouping up the outcomes of these techniques, thus the similar logs would be in clusters and those which are not going to appear in a group would be marked as having potential anomalies.

4. Preprocessing for Other Algorithms:
t-SNE can be used as a preprocessing step for other anomaly detection algorithms, particularly when dealing with high-dimensional data. The transformation of high-dimensional data into lower dimensionality makes it easier for other algorithms to apply techniques like K-Means, Isolation Forest, or autoencoders in order to detect patterns and anomalies.

Example Use Case in Anomaly Log Detection:

For instance, imagine that you have server logs with hundreds of different features, including IP addresses, types of requests, error codes, and timestamps. Then you can preprocess these logs, for example transform categorical data into numeric, and apply t-SNE to reduce the dimensionality of your data. The output would be a 2D plot in which log entries of normal events would tend to bunch, whereas log entries for abnormal events-for example, anomalies such as odd appearing spikes in traffic or errors-may appear as isolated points. Those isolated points may represent anomalous records.

Benefits of Using t-SNE for Anomaly Detection:

1. Visualization of Complex Data:

t-SNE simplifies the visual interpretation of complex, high-dimensional log data, which makes it easier to understand the structure of the data and identify anomalies.

2. Handling Non-Linear Data:

t-SNE is particularly effective in handling non-linear relationships between features, which is common in real-world log data.

3. Non-Linear Separation of Clusters:

Unlike linear techniques (like PCA), t-SNE can capture non-linear relationships in the data, which allows it to identify more complex patterns of anomalies that linear methods might miss.

Limitations of t-SNE:

1. Computationally Expensive:

t-SNE can be computationally expensive, especially when dealing with very large datasets. The algorithm has a time complexity of O(N^2), where N is the number of data points.

2. Doesn't Preserve Global Structure:

Although t-SNE excels at preserving local structures, the global structures become distorted in the sense that the relationship between clusters that are far apart is lost. Therefore, t-SNE is not a suitable tool for general clustering or classification tasks but for visualization and exploratory data analysis purposes.

3. Random Initialization:

t-SNE uses stochastic methods, so the resulting visualization can vary with different runs unless you carefully set the random seed.

Conclusion:

High-dimensional log data can be visualized with outstanding t-SNE, and possible anomaly discoveries can be made by reducing the dimension of the data. Although it is not an algorithm for anomaly detection, projecting complex log data into 2D or 3D space makes it quite useful for discovering patterns and clusters and outliers that can then be marked as suspicious for further analysis. But the approach of t-SNE, combined with other algorithms like K-Means or Isolation Forest, will greatly boost the overall process of anomaly detection in making it easier to detect anomalies and problems in system logs.

3.5 ONE CLASS SVM Project:

One-Class SVM is one kind of specific machine learning algorithm mainly used for anomaly or outlier detection. Being a type of support vector machine, unlike traditional binary SVM that separates two classes, One-Class SVM is trained on only one class assuming it is "normal" data and hence finds an application in unsupervised tasks like anomaly detection. The algorithm works by mapping normal data points into a high-dimensional space and boundary specification around them, often in the form of a hypersphere or hyperplane. In testing, data points falling outside such a boundary are labelled as anomalies, and it has thus far proved effective in identifying data points with significant deviation from typical patterns. One-Class SVM is generally used in applications where anomalies are scarce. For example, fraud detection, network intrusion detection, and system log analysis.

Use Case in Anomaly Detection:

One-Class SVM can be applied to monitor system logs for rare events or errors and flag unusual patterns that could potentially indicate security breaches, failure, or performance degradation. It is useful to detect anomalous behaviours in applications, servers, or network systems wherein only "normal" operational data are available for training. For example, while building an IT infrastructure, One-Class SVM can analyse server application logs or network device logs for possible detection of one or more intruder signs or the start of unknown activity that may potentially break system integrity and compromise security.

Advantages

-One-Class SVM has several advantages, especially in anomaly detection:
-Minimal Data Requirement: It does not need labelled anomalous data, which is often scarce.
-High-Dimensional Flexibility: Handles high-dimensional data well, suitable for complex logs or network data.
-Novelty Detection: Effective at identifying new, previously unseen patterns and outliers in data.
-Hyperparameter Control: Offers parameters like nu and gamma to balance model sensitivity to anomalies, aiding in achieving desired detection thresholds.

Disadvantages:

-However, there are also notable limitations:
Sensitivity to Hyperparameters:
-Choosing improper values for nu and gamma can lead to high false positives or negatives.
-Inefficiency on Imbalanced Data:
Tends to misclassify borderline anomalies as normal when trained solely on a single-class dataset.
-Computational Cost:
Training in high-dimensional spaces can be resource-intensive, potentially limiting scalability.

Efficiency and Accuracy :

One-Class SVM is very efficient for small to moderately sized data volumes but can grow computationally expensive in highly dimensional spaces or very large datasets. Also, its accuracy is very sensitive to parameter tuning; with well-tuned parameters, it can result in fairly reliable prediction, though it hardly happens, especially in highly imbalanced settings where anomalous patterns are closer approximations of normal ones. Despite these limitations, One-Class SVM is a very powerful tool for novel anomaly detection and usually combined with other methods for detecting anomalies for improvement in overall accuracy and robustness in log-based applications.

Conclusion :

Very useful for anomaly detection, especially when only "normal" data are available to train One-Class SVM. Can be applied very widely, for example in log analysis, fraud detection and network security, in detecting infrequent, unusual or irregular events. It highly performs in high-dimensional spaces and is rather flexible for different kinds of data; however, it is very well-known to be sensitive to the hyperparameters and computationally intensive if large datasets are used. However, appropriate tuning of parameters and feature engineering can provide good results in one-class SVM for the identification of novel or unexpected behaviours, making them fit for log anomaly detection projects pretty well. It can be utilized with other techniques to improve the performance of the technique and make a more holistic yet accurate anomaly-detecting system.

Chapter 4

PROTOTYPE

4.1 PROTOTYPE

Prototype Overview:

A prototype for a log anomaly detection project using One-Class SVM can indeed be developed as part of a software-only implementation. This prototype would focus on demonstrating the core functionalities of the anomaly detection system, specifically in log- based applications.

Data Collection and Preprocessing Module :

Actual Design of Prototype The prototype will comprise of a data collection and preprocessing module, which will collect server logs or application logs, parse them into structured data. Feature extraction will be based on pulling in those key attributes within the logs, maybe those event types or timestamps, and normalizing those values for use in the SVM algorithm.

Model Training Module :

Then, a model training module would be configured using One-Class SVM where the algorithm learns a boundary that encapsulates normal data points, based on such features. The model's hyperparameters - nu and gamma- would then be tuned in order to get a really good trade-off between getting the maximum accuracy in detection and having minimum false alarm rate.

Anomaly Detection Phase :

This would evaluate new log entries in real-time or batches during the anomaly detection phase, marking anomalies as points outside the decision boundary learned during training. It raises alarms or generates anomaly reports, which then developers or system administrators can use in further analyses.

User Interface :

Optionally, a simple user interface, either command-line or web-based, could allow users to upload logs, review results, and view summaries of detected anomalies.

Purpose of the Prototype :

This prototype would primarily act as a proof of concept, showing the feasibility and reliability of One-Class SVM in detecting novel or unusual patterns in log data.

Limitations:

Though this first version of the implementation would, by necessity, be devoid of production features like automatic retraining, scaling optimizations, and advanced anomaly scoring, it would lay the groundwork through which stakeholders could evaluate its accuracy, efficiency, and potential.

4.2 SOFTWARE USED :

Programming Language :

In summary, for such a project focused on log anomaly-detection with the use of One-Class SVM, the choice of language is Python: it is, after all, one of the most popular languages concerning ML and data science, highly ideal for implementing and fine-tuning the model, which is remarkably robust in the whole ecosystem.

Data Handling and Manipulation:

For transforming raw log management into structured data, Pandas is the most valuable asset. It ensures easy management and manipulation of data for a smooth transformation of raw logs into a structured data format for conversion into a format fit for analysis and model training.

Log Parsing :

Regular expressions or Log parser library can be used to parse and process log files. For instance, it helps in extracting specific information from the unstructured logs; it is possible to capture key attributes, event types, and timestamps efficiently.

Data Visualization :

Graphics of results can be produced with Matplotlib or Seaborn if visualization of anomalies is required. Visualizations may indicate certain patterns in the data and help to understand the performance by a model.

Development Environment :

For a log anomaly detection project using One-Class SVM, Google Colab and Jupyter Notebook would be the perfect choice because they support interactive coding, hence easy experimentation with data preprocessing, model training, and evaluation in real time. Additionally, Google Colab provides free GPU and TPU support, which can speed up model training, especially for large datasets, and it is cloud-based, making code shareable and collaboratively workable.

Code Versioning and Collaboration :

Very importantly, using Git together with GitHub or GitLab is of immense benefit for versioning and teamwork. It goes ahead to track changes in codes, handle different versions, and also aid teamwork so as to collaborate easily and revert to previous versions of code if necessary.

Integrated Development Environments (IDEs) :

For people who like a more traditional development environment, Visual Studio Code is a quite powerful, versatile IDE that supports development in Python, has integrated Git support, and allows the use of various extensions for data science. VS Code provides a stable workspace with syntax highlighting, debugging tools, and direct integration with Jupyter Notebooks to run code cells interactively.

Prototype Deployment and User Interface:

Prototyping can be handled by simple web-based interfaces that will be deployed or even used for the purpose of creating a user interface, using Flask or Streamlit. These would ease uploading logs, reviewing anomaly reports, and interacting with model output.

4.3LIBRARIES USED

1. Data Preprocessing and Handling :

Data processing is the most critical step in any anomaly detection project. Commonly, libraries like Pandas are used for importing, cleaning, and processing the log file. It facilitates easier processing of large datasets, along with some basic tools like dropna(), groupby(), merge(), and so on, to sort out data accordingly. NumPy is required for doing number calculations on the data, such as normalization or aggregation, and Regex (re) is good when needed to extract some part of information like error codes, timestamps from raw log text using regular expressions.

2. Anomaly Detection and Machine Learning

Log data anomaly detection is impossible without machine learning techniques. Isolation Forest and One-Class SVM algorithms in Scikit-learn, larger, more complex using TensorFlow, and PyTorch for time-series data, where Statsmodels help find trends with the help of PyOD's differentiated abnormality detection methods like KNN and LOF.

3. Natural Language Processing (NLP) Libraries :

That is to say, this only happens if your log data comprises textual content; otherwise, NLP libraries will do much of the processing. NLTK and SpaCy are good choices for text processing, which include tokenization, lemmatization, and removal of stop words. Useful to convert raw log text into meaningful features, these libraries include Genism, which focuses more on topic modelling and similarities within large text datasets - useful in the sense of clustering log entries or detecting unusual patterns.

4. Visualization :

Visualizing log data and the results of anomaly detection is a powerful way to understand patterns. Matplotlib and Seaborn are commonly used for static plotting, creating histograms, scatter plots, and heatmaps to explore the distribution of log data and highlight anomalous entries. For interactive visualizations, Plotly allows the creation of more dynamic charts and dashboards. Additionally, Grafana and Kibana are widely used in real-time log monitoring, providing visual insights into logs, anomalies, and other system metrics.

5. Time-Series Analysis :

Time-stamped logs demand anomaly detection in time-series data. Prophet is the type of library developed by Facebook to model and forecast time-series data about finding seasonal patterns and shifts in trends that probably present anomalies. Scikit-time is a specialization of scikit-learn and more oriented toward time-series data. It can be employed in processing and modelling such data for the detection of anomalies in time series.

6. Log Parsing and Integration:

Often, logs need to be parsed and integrated with other systems. There's Loguru for real-time anomaly detection as well as a flexible Python library for capturing log events. Advanced log processing can be performed using ELK Stack. Logstash handles ingestion and preprocessing of logs, while Elasticsearch ensures efficient searching and indexing of large log datasets.

7. Data Storage and Retrieval:

Log data is frequently stored in databases for easy access and querying. SQLite and PostgreSQL are commonly used for relational databases where logs are stored, allowing for efficient retrieval of log records for anomaly detection. These libraries support querying and manipulating large sets of logs using SQL.

8. Cloud and Big Data Integration:

Large-scale log management, especially a cloud-based setup, works on integration with cloud services and big data tools. The AWS Boto3 is a python library to interact with the APIs provided by the Amazon Web Services, which is helpful for either S3, for storing logs or CloudWatch for real-time stream of logs. PySpark enables distributed computing through Apache Spark for heavy log datasets and anomaly detection algorithms over large datasets in parallelized, efficient terms.

9. Evaluation and Metrics:

An important aspect of performance evaluation, therefore, relates to checking whether the anomaly detection model actually identifies anomalies correctly. Model evaluation tools are provided by scikit-learn, which includes precision, recall, and F1-score metrics for checks. With the help of the use of these metrics, these models can be fine-tuned and meet the required performance standards by practitioners.

4.4 CODE ALGORITHM:

1. Data Loading and Preprocessing :

Loading and preprocessing of log data: It primarily points to loading the data and preprocessing. Normally, the data usually lies in a CSV file, and this is loaded using the libraries. The target variable normally separated from feature would be one that will indicate whether the log entry falls into being either normal or anomalous. Standard Scaler is then applied to ensure that the features are on similar scales so that standardization of the data is made. Models like K-Means and One-Class SVM require standardization because they are sensitive to the scale of their input data.

2. K-Means Clustering :

The K-Means algorithm is applied to the dataset to detect anomalies. As K-Means is a clustering algorithm, it takes the data points and groups them into the predefined number of clusters. Here, the model will be set to create two clusters, one for normal data and one for anomalous data. The algorithm is trained on scaled features and makes predictions on the test set to classify each log entry as part of which cluster it falls into. Logs that are most distant from the centroids of the clusters are assumed to be outliers.

3. t-SNE for Dimensionality Reduction :

Although t-SNE is not that specially designed for anomaly detection, it is a very helpful visualization tool for making the high-dimensional data sets live into two or three dimensions such that one may better understand the structure of data before applying more models. Here, t-SNE is applied to reduce the space of features to just two dimensions; therefore, the visual presentation becomes much easier for exploration of the distribution of normal versus anomalous logs.

4. Isolation Forest :

It is one effective anomaly detection method on high-dimensional datasets. Isolation Forest isolates anomalies simply because it randomly selects features and splits the data, where anomalies are few and distinct. The model is trained on the scaled data and makes predictions on test data to classify log entries as normal or anomalous.

5. One-Class SVM :

Normally, One-Class SVM uses the available normal data in the training set to detect anomalies. In this particular approach, finding a boundary which encloses all the normal data points is most important, such that any point lying outside that boundary is supposed to be an anomaly. In this implementation, One-Class SVM is trained on the normal logs and tested on the full dataset, where the aim is to detect whether any of the unseen log entries deviate from the learned distribution of normal data.

6. Stacking Classifier :

A Stacking Classifier is an ensemble technique that combines multiple base classifiers toward the improvement in performance of the model. In this, Random Forest as well as Naive Bayes are base classifiers while Logistic Regression as a meta-classifier. The base classifiers are trained in an independent manner, after which their outputs have been fed to the meta-classifier for enhancing general anomaly detection.

7. Baseline Classifier :

A Baseline Classifier provides an easy reference point with which to compare the performance of more complex models. In the above example, for instance, Logistic Regression is a baseline classifier. This one is trained over the training set and tested over the test set. It is very basic and serves as a baseline where the much more complicated ensemble methods like Stacking Classifiers or Isolation Forest can be compared to.

8. Ensemble Methods :

In ensemble methods, like Random Forest and Gradient Boosting, the output of many models is combined through the average to make the predictions both more accurate and robust. Two distinct models are trained on the same dataset separately; at the final stage, the outcome of two models is averaged through a simple majority voting technique. Therefore, the ensemble techniques generally outperform the single models of the respective models, especially in noisy data environments like the log anomaly detection task.

9. Model Evaluation :

After training and testing, the models' performance is evaluated using metrics such as precision, recall, and F1-score. These are all about the ability of a model to detect anomalies. The classification report from Scikit-learn is used to display these metrics. Using this approach, you can find the best-performing algorithm or method for deployment for log anomaly detection.

4.5CODE

Anomaly logs Detection Pseudocode:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.cluster import KMeans
from sklearn.manifold import TSNE
from sklearn.ensemble import IsolationForest, StackingClassifier, RandomForestClassifier, GradientBoostingClassifier
from sklearn.svm import OneClassSVM
from sklearn.metrics import classification_report
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.preprocessing import StandardScaler

data = pd.read_csv("logs.csv")
X = data.drop('label', axis=1)
y = data['label']

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3, random_state=42)

kmeans = KMeans(n_clusters=2, random_state=42)
kmeans.fit(X_train)
y_kmeans = kmeans.predict(X_test)

tsne = TSNE(n_components=2, random_state=42)
X_tsne = tsne.fit_transform(X_scaled)

iso_forest = IsolationForest(random_state=42)
iso_forest.fit(X_train)
y_iso_forest = iso_forest.predict(X_test)

one_class_svm = OneClassSVM(nu=0.1, kernel="rbf", gamma="auto")
one_class_svm.fit(X_train)
y_one_class_svm = one_class_svm.predict(X_test)

base_learners = [
   ('rf', RandomForestClassifier(n_estimators=100, random_state=42)),
   ('gnb', GaussianNB())
]
meta_classifier = LogisticRegression()
stacking_clf = StackingClassifier(estimators=base_learners, final_estimator=meta_classifier)
stacking_clf.fit(X_train, y_train)
y_stacking = stacking_clf.predict(X_test)

baseline_clf = LogisticRegression()
```

```
baseline_clf.fit(X_train, y_train)
y_baseline = baseline_clf.predict(X_test)

rf_clf = RandomForestClassifier(n_estimators=100, random_state=42)
gb_clf = GradientBoostingClassifier(n_estimators=100, random_state=42)

rf_clf.fit(X_train, y_train)
gb_clf.fit(X_train, y_train)

y_rf = rf_clf.predict(X_test)
y_gb = gb_clf.predict(X_test)

y_ensemble = np.round((y_rf + y_gb) / 2).astype(int)

print("K-Means Anomaly Detection Results:")
print(classification_report(y_test, y_kmeans))

print("Isolation Forest Anomaly Detection Results:")
print(classification_report(y_test, y_iso_forest))

print("One-Class SVM Anomaly Detection Results:")
print(classification_report(y_test, y_one_class_svm))

print("Stacking Classifier Results:")
print(classification_report(y_test, y_stacking))

print("Baseline Classifier Results:")
print(classification_report(y_test, y_baseline))

print("Ensemble Method (Random Forest + Gradient Boosting) Results:")
print(classification_report(y_test, y_ensemble))
```

Chapter 5

Software Implementation and Results

5.1 PROTOTYPE RESULTS

```
plt.style.use('ggplot')
import matplotlib.style as style
style.use('fivethirtyeight')
plt.rcParams['figure.figsize'] = (15, 8)


from sklearn import preprocessing
from sklearn.decomposition import PCA
import warnings
warnings.filterwarnings('ignore')

df=pd.read_csv("logs_dataset.csv")

df
```

	@timestamp	_id	ip_address
0	July 8th 2019, 14:43:03.000	XswJ0msBoTGddM7vxMDB	10.1.1.285
1	July 8th 2019, 14:43:01.000	dKQJ0msB7mP0GwVzvJjz	10.1.2.389
2	July 8th 2019, 14:42:59.000	CcwJ0msBoTGddM7vtb8y	10.1.1.415
3	July 8th 2019, 14:42:57.000	bKQJ0msB7mP0GwVzrZdT	10.1.1.79
4	July 8th 2019, 14:42:55.000	L6QJ0msB7mP0GwVzpZeI	10.1.1.60
...	...	...	...
721542	June 9th 2019, 00:00:25.000	dkWJOWsBoTGddM7vRUOR	10.1.2.66
721543	June 9th 2019, 00:00:19.000	TBuJOWsB7mP0GwVzLmol	10.1.1.249
721544	June 9th 2019, 00:00:15.000	vUWJOWsBoTGddM7vHkGD	10.1.1.200
721545	June 9th 2019, 00:00:09.000	aUWJOWsBoTGddM7vB0AR	10.1.2.432
721546	June 9th 2019, 00:00:05.000	AEWIOWsBoTGddM7v90Bw	10.1.2.156

721547 rows × 3 columns

Image of full Dataset Set-up

KMeans

```
from sklearn.cluster import KMeans

final_dataset.columns

Index(['ip_address', 'total_count', 'daily_counts',
'is_weekend_ratio',
       'td_mean', 'td_max'],
      dtype='object')
```

Displaying the columns present in the final_dataset

```
requi_feature_cols = ['total_count', 'daily_counts',
        'is_weekend_ratio', 'td_mean', 'td_max']

new_data = final_dataset[requi_feature_cols]

min_max_scaler = preprocessing.MinMaxScaler()

data_scaled = min_max_scaler.fit_transform(new_data)
new_data = pd.DataFrame(data_scaled,columns=requi_feature_cols)

import seaborn as sns
from mpl_toolkits.mplot3d import Axes3D

sns.pairplot(final_dataset[requi_feature_cols])

<seaborn.axisgrid.PairGrid at 0x7c4342ffcc10>
```



```
sns.pairplot(new_data)

<seaborn.axisgrid.PairGrid at 0x7c43370efe20>
```

```
cluster_num = range(1,15)

kmeans = [KMeans(n_clusters=i,
        random_state=RANDOM_STATE).fit(new_data) for i in
        cluster_num]

kmeans

[KMeans(n_clusters=1, random_state=30),
 KMeans(n_clusters=2, random_state=30),
 KMeans(n_clusters=3, random_state=30),
 KMeans(n_clusters=4, random_state=30),
 KMeans(n_clusters=5, random_state=30),
 KMeans(n_clusters=6, random_state=30),
 KMeans(n_clusters=7, random_state=30),
 KMeans(random_state=30),
 KMeans(n_clusters=9, random_state=30),
 KMeans(n_clusters=10, random_state=30),
 KMeans(n_clusters=11, random_state=30),
```

K-means clustering implementation

```
        [-10.478027   ,   4.4418173 ],
        [ -3.2145355 ,  -9.342764  ]], dtype=float32)

final_dataset['tsne-2d-one'] = tsne_result[:,0]
final_dataset['tsne-2d-two'] = tsne_result[:,1]

tsne_cluster = final_dataset.groupby('cluster').agg({'tsne-2d-
        one':'mean','tsne-2d-two':'mean'}).reset_index()

plt.figure(figsize=(16,10))

sns.scatterplot(
    x="tsne-2d-one", y="tsne-2d-two",
    hue="cluster",
    palette=sns.color_palette("hls", 6),
    data=final_dataset,
    legend="full",
    alpha=1
)

plt.scatter(x="tsne-2d-one", y="tsne-2d-two", data=tsne_cluster,
        s=100, c='b')

plt.show()
```



```
centers = kmeans_cluster_model.cluster_centers_

pts = np.asarray(new_data)

pts
```

TSNE Algorithm implementation

Isolated Forest

```
from sklearn.ensemble import IsolationForest

outlier_frac = 0.028

final_dataset.columns

Index(['ip_address', 'total_count', 'daily_counts',
'is_weekend_ratio',
       'td_mean', 'td_max', 'cluster', 'tsne-2d-one', 'tsne-2d-two',
       'sum_square_dist', 'sum_squa_dist', 'anomaly_kmeans'],
      dtype='object')

new_data.head()
```

	total_count	daily_counts	is_weekend_ratio	td_mean	td_max
0	0.038133	0.052632	0.512099	0.884954	0.716621
1	0.502959	0.497076	0.516266	0.231366	0.234332
2	0.044379	0.052632	0.492225	0.863479	0.305177
3	0.025641	0.000000	0.205662	0.919573	0.599455
4	0.042406	0.081871	0.342074	0.872065	0.487738

```
iso_model =  IsolationForest(n_jobs=-1, n_estimators=200,
        max_features=3, random_state=RANDOM_STATE,
        contamination=outlier_frac)

iso_model.fit(new_data)

▾                          IsolationForest

IsolationForest(contamination=0.028, max_features=3, n_estimators=200,
                n_jobs=-1, random_state=30)
```

ISOLATED FOREST

One Class SVM

```
from sklearn.svm import OneClassSVM

svm_model =  OneClassSVM(nu=outlier_frac, degree=2, kernel='rbf')

new_data_clean = new_data.loc[new_data.total_count
        <=new_data.total_count.quantile(1-outlier_frac)]

svm_model.fit(new_data_clean)

▾        OneClassSVM

OneClassSVM(degree=2, nu=0.028)
```



```
final_dataset['anomaly_svm'] = pd.Series(svm_model.predict(new_data))

final_dataset['anomaly_svm'] = final_dataset['anomaly_svm'].map( {1:
        0, -1: 1} )

final_dataset['anomaly_svm'].value_counts()
```

	count
anomaly_svm	
0	367
1	19

ONE-CLASS SVM

```
<Axes: xlabel='tsne-2d-one', ylabel='tsne-2d-two'>
```

```
final_dataset.loc[final_dataset['anomaly_svm']==1]
```

	ip_address	total_count	daily_counts	is_weekend_ratio	td_mean	td_max	cluster	tsn one
9	10.1.1.12	1392	40.0	4.974249	30.152408	250.0	3	-3.8
37	10.1.1.186	1372	37.0	4.965217	30.557987	308.0	3	-2.8
62	10.1.1.249	4301	116.5	6.004886	9.459535	101.0	1	-4.9
118	10.1.1.386	4300	118.5	6.388316	9.453361	104.0	1	-5.0
164	10.1.1.486	4317	117.0	5.611026	9.417285	108.0	1	-4.8
177	10.1.1.63	4339	112.0	6.136513	9.368142	101.0	1	-4.9
183	10.1.1.73	1484	40.5	5.480349	28.191504	195.0	3	-4.2
188	10.1.1.86	4293	113.0	6.276271	9.456897	110.0	1	-5.0
225	10.1.2.178	1400	39.0	7.284024	29.964975	258.0	0	10.
233	10.1.2.195	2859	76.5	6.919668	14.448915	206.0	2	-10
255	10.1.2.249	4353	112.0	5.876777	9.332721	102.0	1	-4.9
282	10.1.2.314	2891	75.0	5.312227	14.274740	131.0	2	-7.9
286	10.1.2.323	1408	43.5	7.045714	29.719261	458.0	5	10.
311	10.1.2.386	4326	108.0	5.910543	9.392370	110.0	1	-4.9
331	10.1.2.432	1437	40.5	6.445596	29.176880	466.0	5	9.7
357	10.1.2.486	4251	114.0	6.316695	9.571059	99.0	1	-5.0
370	10.1.2.63	4372	121.0	6.435374	9.268588	118.0	1	-5.0
373	10.1.2.67	1364	40.0	6.976608	30.748349	229.0	0	9.2
381	10.1.2.86	4307	111.0	6.083882	9.441013	122.0	1	-4.9

Performance of the Algorithm

38640f50a6984bcbb4eb62abcf5a2f77

We must manually categorise the data and examine the performance of the individual algos because it is unsupervised.

```
total_counts_condition = (final_dataset['total_count'] >=
        final_dataset['total_count'].quantile(0.90))
daily_counts_condition =  (final_dataset['daily_counts'] >=
        final_dataset['daily_counts'].quantile(0.90))

final_dataset['anomaly_manual'] = (total_counts_condition |
        daily_counts_condition).astype(int)

final_dataset.head()
```

	ip_address	total_count	daily_counts	is_weekend_ratio	td_mean	td_max	cluster	tsne-one
0	10.1.1.1	1446	40.0	6.194030	28.999308	362.0	5	7.551
1	10.1.1.100	2860	78.0	6.204030	14.427072	185.0	2	-10.2
2	10.1.1.101	1465	40.0	6.146341	28.520492	211.0	4	1.297
3	10.1.1.106	1408	35.5	5.458716	29.771144	319.0	3	-0.46
4	10.1.1.109	1459	42.5	5.786047	28.711934	278.0	3	-0.41

```
from sklearn.metrics import f1_score, roc_auc_score, accuracy_score,
        confusion_matrix

def get_sensitivity_specificity(y_true, y_pred):
    cf = confusion_matrix(y_true, y_pred)
    sensitivity = cf[0,0]/(cf[:,0].sum())
    specificity = cf[1,1]/(cf[:,1].sum())
    return sensitivity, specificity

f1_iso =
        f1_score(final_dataset['anomaly_manual'],final_dataset['anomaly_isolated'])
acc_iso =
        accuracy_score(final_dataset['anomaly_manual'],final_dataset['anomaly_isolated'])
roc_iso =
        roc_auc_score(final_dataset['anomaly_manual'],final_dataset['anomaly_isolated'])
sen_iso, spec_iso =
        get_sensitivity_specificity(final_dataset['anomaly_manual'],final_dataset['anomaly_isolated'])
met_iso = {
           'f1_score': f1_iso,
           'accuracy': acc_iso,
           'roc_score': roc_iso,
           'sensitivity': sen_iso,
           'specificity': spec_iso
          }

f1_kmeans =
        f1_score(final_dataset['anomaly_manual'],final_dataset['anomaly_kmeans'])
acc_kmeans =
        accuracy_score(final_dataset['anomaly_manual'],final_dataset['anomaly_kmeans'])
roc_kmeans =
        roc_auc_score(final_dataset['anomaly_manual'],final_dataset['anomaly_kmeans'])
sen_kmeans, spec_kmeans =
        get_sensitivity_specificity(final_dataset['anomaly_manual'],final_dataset['anomaly_kmeans'])
met_kmeans = {
           'f1_score': f1_kmeans,
           'accuracy': acc_kmeans,
           'roc_score': roc_kmeans,
           'sensitivity': sen_kmeans,
           'specificity': spec_kmeans
          }

f1_svm =
        f1_score(final_dataset['anomaly_manual'],final_dataset['anomaly_svm'])
acc_svm =
        accuracy_score(final_dataset['anomaly_manual'],final_dataset['anomaly_svm'])
roc_svm =
        roc_auc_score(final_dataset['anomaly_manual'],final_dataset['anomaly_svm'])
```

38640f50a6984bcbb4eb62abcf5a2f77

```
sen_svm, spec_svm =
        get_sensitivity_specificity(final_dataset['anomaly_manual'],final_dataset['anomaly_svm'])
met_svm = {
            'f1_score': f1_svm,
            'accuracy': acc_svm,
            'roc_score': roc_svm,
            'sensitivity': sen_svm,
            'specificity': spec_svm
           }

metrics = {'isolated_forest': met_iso,
           'kmeans': met_kmeans,
           'svm': met_svm
           }

all_metrics = pd.DataFrame.from_dict(metrics)

all_metrics
```

	isolated_forest	kmeans	svm
f1_score	0.303030	0.289855	0.270270
accuracy	0.880829	0.873057	0.860104
roc_score	0.589399	0.584867	0.577314
sensitivity	0.880000	0.879032	0.877384
specificity	0.909091	0.714286	0.526316

Ensemble models for Classification

```
from sklearn.preprocessing import LabelEncoder
final_dataset['ip_address'] =
        LabelEncoder().fit_transform(final_dataset.iloc[:, -1])

features = ['ip_address', 'total_count', 'daily_counts',
        'is_weekend_ratio', 'td_mean', 'td_max']
X = final_dataset[features]
y = final_dataset['anomaly_isolated']

X, y = final_dataset[features], final_dataset['anomaly_isolated']
X.shape, y.shape

((386, 6), (386,))
```

ENSEMBLE TECHNIQUES USED IN CODE

Baseline Classifier

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.multiclass import OneVsRestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import StackingClassifier
from sklearn.linear_model import LogisticRegression

from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold

rf = OneVsRestClassifier(estimator = GaussianNB())

# Create RepeatedStratifiedKFold cross-validator with 10 folds, 3
        repeats and a seed of 1.
```

ı/Local/Temp/be71e89b-a5ce-4ceb-a592-0dda9d4afdd3_vertopal.com_Anomaly_Log_Detection_PJT[1].zip.dd3/45b

38640f50a6984bcbb4eb62abcf5a2f77

```
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3,
        random_state=20)

# Calculating accuracy using `cross_val_score()` with model
        instantiated, data to fit,
# target variable, 'accuracy' scoring, cross validator, n_jobs=-1, and
        error_score set to 'raise'
n_scores = cross_val_score(rf, X, y, scoring='f1_weighted', cv=cv,
        error_score='raise')


# Print mean and standard deviation of n_scores:
print(n_scores)
print('Baseline scores \n mean f1 weighted: %.3f with a %.3f standard
        deviation in scores ' % (np.mean(n_scores),
        np.std(n_scores)))
```

```
[0.91306815 0.92877493 0.86485671 0.94457323 0.96084546 0.95655271
 0.96070175 0.91064618 0.91064618 0.87780396 0.92877493 0.83100233
 0.94457323 0.91306815 0.92877493 0.95655271 0.94306894 0.87780396
 0.94306894 0.96070175 0.96084546 0.91306815 0.91306815 0.89725275
 0.91306815 0.86824618 0.89434985 1.         0.94306894 0.96070175]
Baseline scores
 mean f1 weighted: 0.924 with a 0.036 standard deviation in scores
```

Stacking Classifier

Stacking classifier takes:

1. estimators: List of baseline classifiers
2. final_estimator: Defined meta classifier
3. cv: Number of cross validations to perform

```
def create_stacking_models():
    base_models = list()
    base_models.append(('KNNC', KNeighborsClassifier(n_neighbors =
        len(np.unique(y))
                                                     , weights =
        'distance')
                       ))
    base_models.append(('SVC', SVC(kernel = 'linear'
                                   , class_weight = 'balanced'
                                   , break_ties = True)
                       ))
    base_models.append(('GNB', GaussianNB()))
    base_models.append(('RF', RandomForestClassifier(n_estimators=
        200,
                                                     oob_score = True,
                                                     class_weight =
        "balanced",
                                                     random_state = 20,
                                                     ccp_alpha = 0.1)
                       ))


    meta_model = LogisticRegression()
    final_model = StackingClassifier(estimators = base_models, ##Base
        estimators which will be stacked together
                                     final_estimator = meta_model,
                                     cv = 5
                                     )
    return final_model

def models_all():
    all_models = dict()
    all_models['KNNC']= KNeighborsClassifier(n_neighbors =
        len(np.unique(y))



                                         , weights = 'distance')
    all_models['SVC']= SVC(kernel = 'linear'
                           , class_weight = 'balanced'
                           , break_ties = True
                           )
    all_models['RF']= RandomForestClassifier(n_estimators= 200,
                                             oob_score = True,
                                             class_weight =
        "balanced",
                                             random_state = 20,
                                             ccp_alpha = 0.15)
    all_models['GNB'] = GaussianNB()
    all_models['Stacking'] = create_stacking_models()
    return all_models

def evaluate_model(model):
    cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3,
        random_state=42)
    scores = cross_val_score(model, X, y, scoring='f1_weighted',
        cv=cv, error_score='raise')
    return scores

model_results = list()
models = models_all()
names = list()

# Creating a for loop that iterates over each name, model in models
      dictionary
for name, model in models.items():
    scores = evaluate_model(model)
    model_results.append(scores)
    names.append(name)
#   print(model_results)
    print('>%s %.3f (%.3f) \n' % (name, np.mean(scores),
        np.std(scores)))
```

```
>KNNC 0.994 (0.012)

>SVC 0.996 (0.011)

>RF 0.996 (0.011)

>GNB 0.921 (0.035)

>Stacking 0.996 (0.011)
```

```
# Flatten the data for the boxplot
flat_names = np.repeat(names, [len(r) for r in model_results])
flat_results = np.concatenate(model_results)

# Create the boxplot with reshaped data
plt.figure(figsize=(15,5))
sns.boxplot(x=flat_names, y=flat_results, showmeans=True)
plt.title("Model Comparisons")
plt.show()
```

Conclusion

- With a stacking model, we were able to get an F1 score of 96.7%, which was greater than the target of 81.1%

Chapter 6

Conclusion

6.1CONCLUSION

Anomaly detection in log data was accomplished by using several types of machine learning models. The detected anomalies can be used in security breach, system failure, and sometimes abnormal behaviour in software systems. Many algorithms have been tested, each contributing differently to the task of anomaly detection.

K-Means Clustering :

K-Means clustering was used to cluster the log entries: Anomalies will be shown as outliers based on how far away, they lie from the centroids of the clusters. While K-Means did help identify some very basic anomalies, it was highly dependent on the number of clusters selected and the algorithm cannot easily pick up more sophisticated patterns inside that data.

T-SNE for Visualization:

Although t-SNE didn't take part actively in anomaly detection, it helped with the process of dimensionality reduction and therefore visualized the data in a 2D presentation. Visual checking as in this manner proved helpful in understanding how the normal versus anomalous logs have been aligned hence further comprehension of the data distribution. However, t-SNE isn't a classification algorithm hence it should be used merely for visualization purposes.

Isolation Forest :

The Isolation Forest method was found to be very effective for anomaly detection. The anomalies are isolated in this method by selecting features and creating partitions randomly. This method is highly efficient in a high-dimensional space. Even on unbalanced data, the model performed well which makes this model appropriate for rare and outlying log entries.

One-Class SVM:

One-Class SVM was trained to model the normal log entry distribution in order to find outliers in the data set. This worked well when only normal data was available for training, whereas in more diverse data sets, it has its own limitation. Its ability to model complex decision boundaries helped to outperform other models in anomaly detection where others might have failed.

Stacking Classifier :

Utilizing a Stacking Classifier with a mixture of base classifiers using Random Forest and Naive Bayes with a Logistic Regression meta-classifier greatly improved the precision level of detection. Pool the strengths of a number of algorithms; the results from the Stacking classifier were very robust, an illustration of how ensembles often outperform their individual constituents for challenging tasks such as anomaly detection.

Baseline Classifier :

The Logistic Regression model has been the baseline classifier, providing an easy comparison on a simpler basis for the more complex ones. Though less complex, it would actually be helpful in appraising the performance of other techniques and providing a quick and efficient solution for anomaly detection.

Ensemble Methods:

Improving further anomaly detection was done using ensemble methods, the combination of Random Forest and Gradient Boosting, in aggregating predictions of multiple models. In this majority voting, results are more accurate and reliable than individual classifiers, hence the power of ensemble learning in the improvement of anomaly detection results.

REFERENCES

[1] **Qia et al. (2022)** - *Adanomaly: Adaptive Anomaly Detection for System Logs with Adversarial Learning*
Conference**:** Proceedings of the 2022 ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD).

[2] **Saygılı et al. (2024)** - *Anomaly Detection on Servers Using Log Analysis*
Conference**:** To be presented at the 2024 IEEE International Conference on Big Data (BigData).

[3] **Zhang et al. (2022)** - *LogST: Log Semi-supervised Anomaly Detection Based on Sentence-BERT*
Conference**:** 2022 IEEE International Conference on Data Mining (ICDM).

[4] **Cao et al. (2017)** - *Machine Learning to Detect Anomalies in Web Log Analysis*
Conference**:** IEEE International Conference on Machine Learning and Applications (ICMLA).

[5] **Le & Zhang (2021)** - *Log-based Anomaly Detection Without Log Parsing*
Conference**:** 2021 IEEE International Symposium on Software Reliability Engineering (ISSRE).

[6] **Pithode & Patheja (2022)** - *A Study on Log Anomaly Detection using Deep Learning Techniques*
Conference**:** 2022 International Conference on Advances in Computing, Communication, and Data Science (CCDS).

[7] **Majd et al. (2022)** - *A Comprehensive Review of Anomaly Detection in Web Logs*
Conference: 2022 ACM International Conference on Web Search and Data Mining (WSDM).

[8] **Xu, W., Huang, L., Fox, A., Patterson, D., & Jordan, M. I. (2009)** - *Detecting Large-Scale System Problems by Mining Console Logs*
Conference**:** Proceedings of the ACM Symposium on Operating Systems Principles (SOSP).

[9] **Du, M., & Li, F. (2016)** - *Spell: Streaming Parsing of System Event Logs*
Conference**:** 2016 IEEE International Conference on Data Engineering (ICDE).

[10] **Kim, Y., Cummins, J., & Abraham, T. (2018)** - *Log-based Predictive Maintenance for Cyber-Physical Systems Using Machine Learning*
Conference**:** 2018 ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS).

[11] **Zhang, J., Chen, Z., Lin, Y., Liu, Z., & Zhang, H. (2019)** - *A Deep Learning Approach for Network Log Anomaly Detection*
Conference**:** 2019 International Joint Conference on Neural Networks (IJCNN).

[12] **He, S., Zhu, J., He, P., & Lyu, M. R. (2017)** - *Log-based Anomaly Detection without Log Parsing*
Conference**:** 2017 IEEE International Conference on Software Engineering (ICSE).

[13] **Kavousi-Fard, A., & Samet, H. (2021)** - *Anomaly Detection in Smart Grid Data Using Hybrid Machine Learning Models*
Conference**:** 2021 IEEE Power & Energy Society General Meeting (PESGM).