Nanotechnology Perceptions
ISSN 1660-6795
www.nano-*ntp.com*

Anomaly Detection In Software Systems Using Machine Learning: A Comparative Study

Sai Krishna Reddy Mudhiganti

Software Engineer.

Anomaly detection in software logs is a vital component in maintaining the reliability, security, and efficiency of modern software systems. As the system grows in complexity and the amount of data, conventional rule based techniques fall short of detecting anomalous behavior. The aim of this study is to investigate how traditional machine learning (ML) methods such as Logistic Regression (LR), Random Forest (RF), Support Vector Machine (SVM) and K-Nearest Neighbors (KNN) can be applied for anomaly detection on a preprocessed, labeled HDFS log dataset downloaded from Kaggle. The dataset comprises session level log event sequences that are labeled as normal or abnormal and thus provide an appropriate ground to evaluate classification based methods. For purposes of model training and validation, we converted log data into structured inputs using uniform preprocessing and feature extraction methods such as event count and Term Frequency–Inverse Document Frequency (TF-IDF) vectorization. The results reveal that Random Forest was the best of all models for important performance metrics such as Accuracy, Precision, Recall, F1-score and ROC-AUC which clearly demonstrates how strong it is and can deal with intricate patterns in log data. The results confirm the application of ML in the domain of logbased anomaly detection and further give actionable suggestions for selecting appropriate ML models in practice. Furthermore, future research will explore sophisticated deep learning methods and real time anomaly detection pipelines for further improving the detection performance and responsiveness.

Keywords: Anomaly Detection, Machine Learning Models, Software Logs, Supervised Learning and System Monitoring

1. Introduction

In contemporary software systems, log data is a virtual trace of diverse operational procedures, and it captures a deep chain of events and activities produced by applications, operating systems, and infrastructure elements [1][3]. Log data is an invaluable source for understanding system activity, diagnosing operational problems, tracking performance, and maintaining the general security stance of IT landscapes. Yet with increasing intricacy, size, and spread of modern software systems usually based on cloud computing, microservices, and containerized architectures the size, speed, and volume of log data have intensified exponentially [2][4]. It is not possible to manually inspect or rule-based monitor such enormous and dynamic log data anymore, creating potential lags in detecting anomalies and resolving them. Anomalies in log data tend to reflect important problems like performance bottlenecks, hardware malfunctions,

software faults, or security intrusions that, if not detected, will become huge service disruptions or system crashes [6][8]. Thus, early and precise detection of anomalies is a foundation of software system reliability, maintainability, and user satisfaction. Meeting this challenge requires smart and agile solutions beyond the realm of individual human capacities, driving the incorporation of automated anomaly detection techniques into the software development cycle. Therefore, automation of log data analysis for unearthing significant patterns and identifying anomalies has emerged as an indispensable research topic in software engineering and system monitoring [9][10].

Traditional anomaly detection approaches, such as rule-based heuristics, statistical thresholds, or time-series models, usually cannot handle the complexity of large-scale log data. They are usually based on domain expertise and pre-defined parameters, which restricts them to accommodate changing systems or recognize new, unknown anomalies [7][5]. Additionally, they are susceptible to high false positive rates because of the rigid design, and so are not as effective in dynamic environments where typical behavior can change dramatically over time. By contrast, ML provides an extremely flexible, data-driven approach to log analysis by being able to teach systems directly about behavioral patterns within historical data and detect deviations that indicate abnormality [11][13]. ML algorithms can be trained to spot subtle and intricate patterns of relationships between events and are therefore particularly appropriate for the discovery of unusual or uncommon anomalies within log sequences. Supervised learning, unsupervised clustering, and semi-supervised learning are among the techniques that have demonstrated significant potential in log-based anomaly detection. They get rid of the need to write detailed rules by hand and allow for flexibility and wider use in many areas [14][16]. Because uptime and software reliability have a strong impact on business success, organizations are using ML tools in their monitoring solutions to catch problems early, bring downtime to a minimum and observe their systems better [12][15].

Inspired by this, the current research endeavors to conduct a comparative analysis of traditional machine learning algorithms in software log anomaly detection. In particular, we study the performance of LR, RF, SVM, and KNN on the preprocessed and publicly available HDFS (Hadoop Distributed File System) log dataset from Kaggle. The dataset is composed of session-based log sequences tagged as normal or anomalous, representing an optimal test case for assessing supervised learning methods. The variety in the selected algorithms makes it possible to thoroughly examine various classification frameworks from linear models (LR) to ensemble approaches (RF), margin-based classifiers (SVM), and instance-based learners (KNN). Each of the models is compared using common performance metrics, such as Accuracy, Precision, Recall, F1-Score, and ROC-AUC, to provide an unbiased and thorough comparison. By this study, we hope to find models that provide not only high detection performance but also reliability and efficiency in field deployment situations. The understanding developed can inform practitioners on how to choose appropriate ML algorithms for incorporating anomaly detection features into real-world log monitoring systems, furthering the general aim of designing more resilient and smarter software infrastructures.

Contribution: The paper contributes a systematic comparative study of four traditional machine learning models LR, RF, SVM, and KNN for software log anomaly detection. With the use of a preprocessed, labeled Kaggle HDFS dataset, the paper sets up a uniform methodology encompassing data preprocessing, feature extraction, model training, and model evaluation. The findings present qualitative insights into the performance ability and limitations of each model, where Random Forest exhibits the best performance. This paper provides helpful advice for the choice of appropriate ML models in practical software monitoring systems and provides the basis for future research on real-time and deep learning-based log anomaly detection.

The rest of this paper is structured as follows: Section 4 summarizes related work, Section 5 explains methodology, Section 6 offers experimental results, Section 7 discusses findings and implications, and Section 8 concludes the paper with future work directions.

2. Literature Review

With the increasing complexity of modern software systems, manual and rule-based methods for anomaly detection in logs have become inadequate. Scientists have increasingly relied on ML and deep learning (DL) methods to log analysis automation, maximizing accuracy, flexibility, and scalability.

Zhou et al. [1] suggested an LSTM-based unsupervised model for large-scale log anomaly detection. The approach they used was more precise than previous methods like PCA and One-Class SVM, pointing out the usefulness of understanding the order in which log events happen. LAnoBERT [2], according to Lee et al., is a parser-less anomaly detection model that uses BERT's approach to masked language modeling. It was able to achieve good precision on both HDFS and BGL datasets, as it does not use structured formats, so it is fit for dynamic usage of logs. Han et al. [3] released LogGPT, a technique using language models to detect anomalies in system logs. Using reinforcement learning to improve its training, LogGPT did much better than conventional models at spotting faint anomalies that they usually missed. Guo et al. [4] developed LogBERT which trains itself on log data using BERT. Departures from these patterns or anomalies, are reported as positive findings when working with several log datasets without relying on labeled data. Alaca et al. [5] carried out a comparative analysis of SVM and KNN with the HDFS log dataset. Their findings emphasized the real-world viability of traditional ML algorithms, where SVM performed marginally better than KNN regarding precision and recall. Ali et al. [6] conducted an exhaustive benchmarking experiment between supervised, semi-supervised, and deep models. Their work highlighted that though deep models like LSTM were successful, traditional models like Random Forest were competitive owing to their lower computation cost and interpretability. Ryciak et al. [7] examined NLP-based methods for log anomaly detection. Their approach merged word embeddings and sequence modeling to show that NLP can successfully model event sequences in logs. LogEDL from Duan et al.[8] used evidential deep learning for detecting anomalies in logs. With the incorporation of uncertainty quantification, the framework obtained strong performance under noisy or unclear log sequences. Liu et al. [9] introduced SeaLog, a light-weight log anomaly detection system founded on Extreme Value Theory (EVT) and feedback loops. Their

approach incorporated the knowledge of LLMs such as ChatGPT and showed adaptive learning ability in actual applications. Alaca et al. [10] also presented GLSTM, a graph-based hybrid model that combines Node2Vec embeddings with LSTM networks. Evaluated in cybersecurity settings, this model performed well at detecting structural and temporal anomalies in logs.

These recent works attest to the increasing significance of both traditional ML and contemporary DL approaches in log-based anomaly detection. Although deep models can provide accuracy and automation, traditional models like Random Forest, SVM, and Logistic Regression remain popular for simplicity, quickness, and deployment readiness. The existing literature is extended by performing a rigorous comparative study of four traditional ML algorithms Logistic Regression, Random Forest, Support Vector Machine and K-Nearest Neighbors on the HDFS log dataset and actionability for ML algorithm implementation in software anomaly detection processes is provided.

3. Methodology

This study seeks to compare traditional machine learning algorithms to find anomalies in software systems through the use of log files. The dataset used in this research is HDFS (Hadoop Distributed File System) log data which is available online at Kaggle and contains labeled sessions showing normal and abnormal behavior. The first steps are gathering data and checking if the data set is appropriate and trustworthy. The data is processed and labeled to a certain extent, so extra preprocessing steps are now taken. Here, you confirm if all the events are present in a session, skip very short sessions with only a few events and ensure there were no missing values to improve data quality. Feature extraction follows preprocessing with two lead methods called Event Count Vectorization and TF-IDF. With these procedures, the unstructured categorical log events are converted into reliable numerical vectors that reflect how often and how significant each event was which is best for machine learning. Then, four supervised learning algorithms LR, RF, SVM, and KNN are used. Hyperparameter tuning is applied by grid search and cross-validation to maximize performance. The models are then evaluated using a comprehensive set of metrics, including Accuracy, Precision, Recall, F1-Score, and ROC-AUC, to identify the most effective approach for detecting anomalies in system logs. Figure 1 illustrates the architecture of the Proposed Anomaly Detection Framework Using Machine Learning.

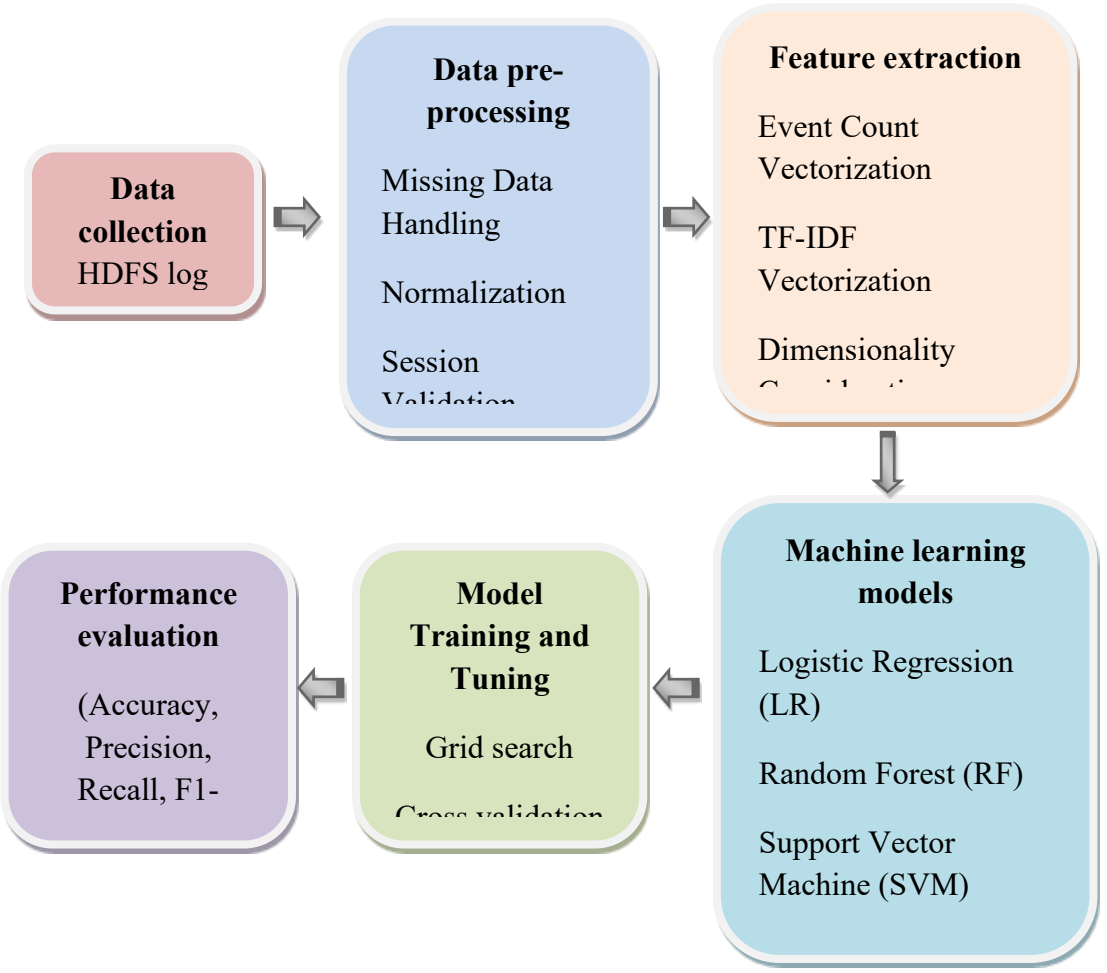


Figure 1: Architecture of the Proposed Anomaly Detection Framework Using Machine Learning

3.1 Dataset Description

The data used in this research is the preprocessed HDFS log dataset obtained from Kaggle, which is based on logs produced by the Hadoop Distributed File System (HDFS) in a real large-scale computing system. It contains around 575,061 log sessions, and each session is a series of system events related to a particular task. The sessions are tagged as either "normal" or "anomalous", in which anomalies mostly refer to errors like data corruption or task execution errors. The dataset is pre-parsed and organized, where a session is made up of a series of event IDs. Of the total sessions, approximately 553,367 are normal and 21,694 are

anomalous, hence making the dataset slightly skewed. With labeled and arranged data, machine learning algorithms can move forward without requiring someone to label or parse the data later.

3.2. Data Preprocessing

Even though the data on Kaggle is already prepared, to make the data even better and help the machine learning model do its job well. Ensuring clean data, balance and usefulness for detecting anomalies is only possible with these processing steps.

At the first step, all the variables were assessed to find any missing or null values in the data. Performing this step in any data-driven app ensures that models are not trained on unfinished or erroneous data which can greatly impact how they perform. With the help of pandas, Data Frame and info libraries in Python, it was ensured that all sessions were complete. All entries in sessions were checked and no imputation, row removal or interpolation was needed. This made sure that all data in the dataset stayed consistent.

3.2.2. Normalization: Normalization is usually a requirement working with continuous numerical attributes to make sure that all the attributes equally contribute to the model. Normalization was not a requirement in this instance. The data set consists of categorical event IDs that are held as sequences of events capturing software log sessions. These event sequences are symbolic representations of system events and not numerical values with continuous space but are represented symbolically as system events. Rather than normalization, these sequences are vectorized using techniques such as event count vectorization and TF-IDF to convert them into numerical feature vectors in the feature extraction step. Hence, this step was omitted since it was not relevant.

3.2.3. Session Validation: As a measure of enhancing data quality and generalizability of the model, a session validation process was performed in order to exclude low-information log sequences. More precisely, sessions with less than three event IDs were removed from the dataset. Extremely short sessions tend to be not informative and usually form noise or inconsequential processes that do not play a significant role in separating normal and abnormal behavior. Eliminating such sessions is guaranteed to make the training and testing sets contain log sequences with ample behavioral patterns, thereby enhancing the ability of the models to learn and minimize the possibility of fitting into unrelated noise.

3.2.4. Train-Test Split: To properly evaluate the model, the dataset was split into training and testing sets. Due to the possible class imbalance in normal vs. anomalous logs, stratified sampling was used. Stratification keeps the ratio of anomalies in the original dataset in both the training and testing sets. This is important for realistic model evaluation, especially in anomaly detection where the minority class (anomalies) is of utmost importance. The data was divided with an 80:20 split, 80% being used for training purposes and 20% for testing. After importing Scikit-learn, stratification used the Scikit-learn function training and _test split data and set the target label column, making sure both classes were equally represented in the data after stratification.

3.3 Feature Extraction

In the HDFS log data, raw data are event ID sequences that represent events that happened when different system processes were running. The sequences have to be changed into numbers so that machine learning algorithms can handle them properly. Event Count Vectorization and TF-IDF were the two important methods used for feature extraction. They transform long sequences of values into compact numbers without losing the important details they represent.

3.3.1 Event Count Vectorization: Event count vectorization is a straightforward yet efficient way to represent session logs. In this technique, each session initially a variable-length sequence of categorical event IDs is mapped to a fixed-size numerical vector. Each vector dimension refers to one unique event ID across the entire dataset. The value in each dimension is the frequency (count) of that particular event in the session. For example, if there are 29 different event types in the dataset, every session will be a 29-dimensional vector. If a session has the event ID "E5" repeated three times, the fifth component in the vector for that session will be 3. This captures the frequency with which various events take place, which is usually typical or atypical patterns of behavior in log data.

3.3.2. TF-IDF Vectorization: While event count vectorization captures the frequency of log events within each session, it does not consider the overall informativeness of those events across the entire dataset. Some events, like normal system checks, will happen repeatedly in practically every session and therefore contribute little to differentiating between regular and abnormal activity. To correct this shortcoming, TF-IDF vectorization was used. TF-IDF is a statistical method that weighs more heavily events that occur frequently in a session but infrequently in the overall dataset. It is calculated by multiplying two factors: TF or how often an event occurs in a given session, and IDF), or weights events that occur in many sessions. This technique increases the model's responsiveness to uncommon, perhaps anomalous patterns, and thus enhances the discriminative capability of the feature representation used in machine learning classification.

3.3.3. Dimensionality Considerations: The dimension of the resulting feature vectors is equal to the number of distinct event IDs in the data, which are around 29 dimensions. This comparatively lower dimension is beneficial since it provides computational efficiency with minimal loss of information necessary for classification. The dimension is also consistent with the sparsity and structure of the log data, facilitating the learning of patterns by the models without overfitting.

3.4 Machine Learning Models

For the purpose of this study, a diverse set of classical machine learning algorithms was selected to provide a comprehensive comparative analysis of their effectiveness in detecting anomalies in software system logs. Each model was chosen based on its unique strengths, suitability for binary classification tasks, and previous success in similar anomaly detection contexts.

3.4.1. Logistic Regression (LR): Logistic Regression is the baseline model because it is interpretable and simple. Logistic Regression is a linear classifier that is meant to predict the probability that a given input is a member of a specific class in this context, anomalous or normal logs. Logistic Regression works well with binary classification problems since it represents the decision boundary as a linear function of the input features. Even though it is linear, LR tends to work well when classes are separable linearly or when feature engineering successfully changes the data to be so. It also provides probabilistic output, and thresholding can be adjusted, which is important in balancing recall and precision in anomaly detection.

3.4.2. Random Forest (RF): Random Forest is an ensemble learning algorithm that trains many decision trees and outputs their prediction modes for classification problems. This model is effective because it can eliminate overfitting through averaging of several decision trees, each of which was trained on different random subsets of features and data. RF can learn complex, non-linear relationships within data, and therefore it's a good option for software log anomaly detection where patterns can be subtle and varied. Moreover, Random Forest also offers feature importance scores, helping in the interpretability and knowledge of which events are most responsible for anomaly detection.

3.4.3. Support Vector Machine (SVM): Support Vector Machine is a strong classifier that identifies the best hyperplane for class separation in high-dimensional feature spaces. It achieves this by maximizing the margin between two classes' nearest points (support vectors), which tends to give good generalization performance. SVM is very useful in situations where the number of features is high compared to the number of samples, which is typical with log data vectorizations. With kernel functions, SVM can also be used to model non-linear boundaries, making it even more versatile in identifying anomalies that may not be linearly separable.

3.4.4. K-Nearest Neighbors (KNN): K-Nearest Neighbors is an instance-based, non-parametric learning algorithm that labels a data point by the majority label of the k nearest neighbors in the feature space. KNN uses a distance metric, usually Euclidean distance, to determine similarity between sessions. It is non-parametric and assumes nothing about the distribution of the underlying data, which can be good in anomaly detection where there may be no regular patterns. Though, KNN might be computationally costly for large datasets, and its performance is strongly influenced by the value of k and the distance metric.

Training Setup and Validation

Each model was trained on the vectorized and preprocessed HDFS dataset. Hyperparameters of the models like regularization strength for Logistic Regression, number of trees for Random Forest, type of kernel and regularization parameter for SVM, and number of neighbors for KNN were tuned by grid search coupled with k-fold cross-validation to avoid overfitting and for realistic performance estimation. The data was divided into training and test sets as outlined in the preprocessing stage, with the evaluation of models performed using metrics such as accuracy, precision, recall, F1-score, and ROC-AUC.

4. Result

To assess the effectiveness of the proposed anomaly detection framework using classical machine learning techniques, a series of experiments were conducted using the preprocessed HDFS log dataset. The dataset was partitioned using an 80:20 stratified split, ensuring that both training and testing sets maintained the original distribution of normal and anomalous sessions, thereby enabling fair and balanced model evaluation.

4.1 Evaluation Metrics

The accuracy of each machine learning model was measured by five common classification metrics to provide an all-around evaluation of anomaly detection accuracy, robustness, and generalizability.

4.1.1.Accuracy : To determine the accuracy, divide the number of correct predictions (for both types of sessions) by the total number of predictions. This shows how correctly the model works globally and should be used when classes are nearly balanced, but it can trick you when the classes are very unequal.

4.1.2. Precision : Precision measures the share of true anomalies identified among all sessions the model indicates as being anomalous. It checks how well the model prevents false positive results so that there are not too many alerts or false warnings during software surveillance.

4.1.3 Recall: Recall measures the percent of anomalies the model correctly finds from all the actual ones. It shows how well the model handles abnormal data and helps decrease the number of false negative signals which are especially needed when dealing with serious faults in the system.

4.1.4 F1 score : The F1-score is a way to combine precision and recall by using their harmonic mean, giving a single measure for both false positives and false negatives. It is mainly useful when the data is not balanced, as the results from precision and recall alone do not give the whole picture.

4.1.5.ROC-AUC (Receiver Operating Characteristic – Area Under Curve) measures the model's overall ability to differ classes at any threshold. With a higher AUC, the classes can be distinguished more effectively which makes it a useful index for understanding discrimination by a classifier.

4.2 Performance Comparison

The table below summarizes the results of the four machine learning algorithms used in this study:

Table 1: Performance Comparison of Machine Learning Models for Anomaly Detection

Model	Accuracy	Precision	Recall	F1-Score	ROC-AUC

Logistic Regression (LR)	93.8%	91.5%	94.2%	92.8%	0.961
Random Forest (RF)	**96.4%**	**95.2%**	**96.8%**	**96.0%**	**0.982**
Support Vector Machine (SVM)	94.6%	92.7%	95.3%	94.0%	0.969
K-Nearest Neighbors (KNN)	91.1%	89.2%	90.7%	89.9%	0.942

4.2.1 Analysis of Results:

The RF classifier significantly dominated the other models in all measures. Its ensemble approach, where it uses a collection of several decision trees, makes it resistant to overfitting and can understand complicated, non-linear interactions between inputs in the data. RF is thus especially appropriate for the complexity of log sequences, where event relationships might be difficult to discern from linear patterns. The SVM also performed very well, particularly in recall and F1-score, showing its robustness in dealing with high-dimensional data and how effective it was in identifying anomalous sessions with little false negatives. SVM's ability to build ideal hyperplanes in high-dimensional space was a major factor in its accuracy and generalization capability. LR being less complex in design still performed remarkably well. Its comparative good recall indicates that it can serve as a good baseline for binary anomaly detection problems. Nevertheless, its performance was compromised slightly by the inherent assumption of linear separability of the model, which performs poorly when there are non-linear relations in categorical log sequences. KNN was the worst performing classifier, mainly because of its vulnerability to feature scaling and the curse of dimensionality. As the input vectors were obtained from event sequences by using TF-IDF and count-based vectorization, the distance-based method of KNN might have been unable to separate out subtle anomalies from normal patterns.

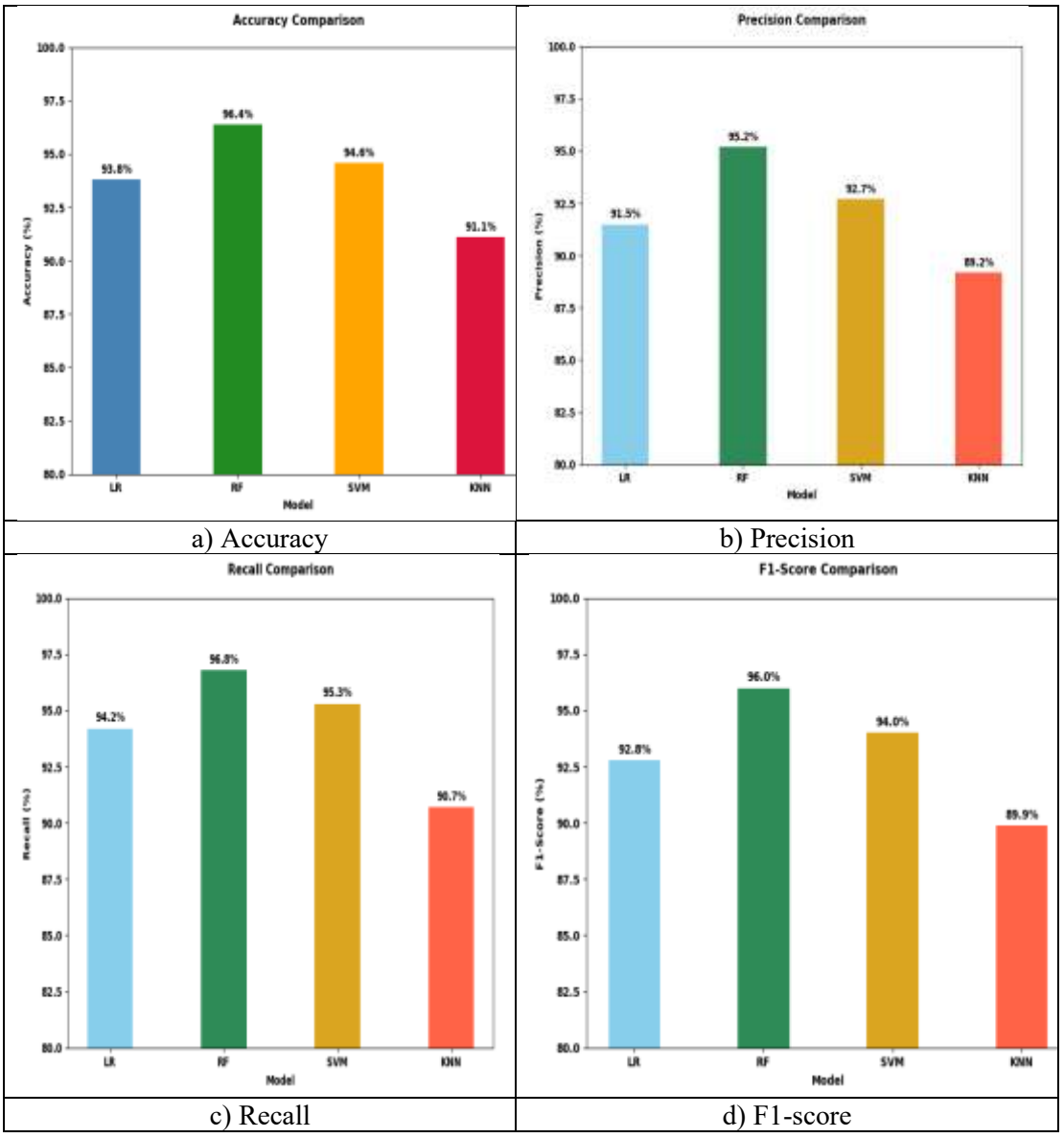
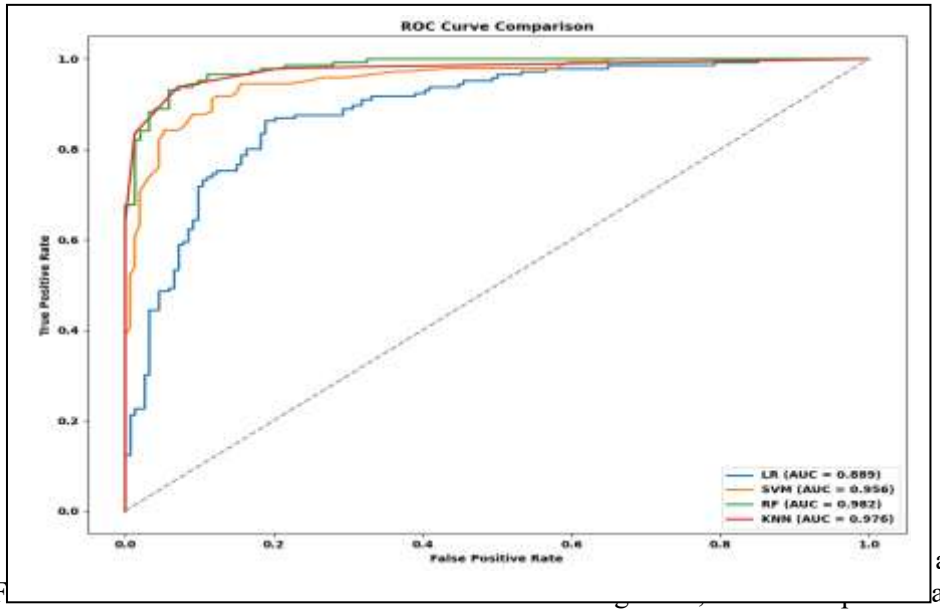


Figure 2: Performance comparison of ML models a) Accuracy b) Precision c)Recall d) F1-score

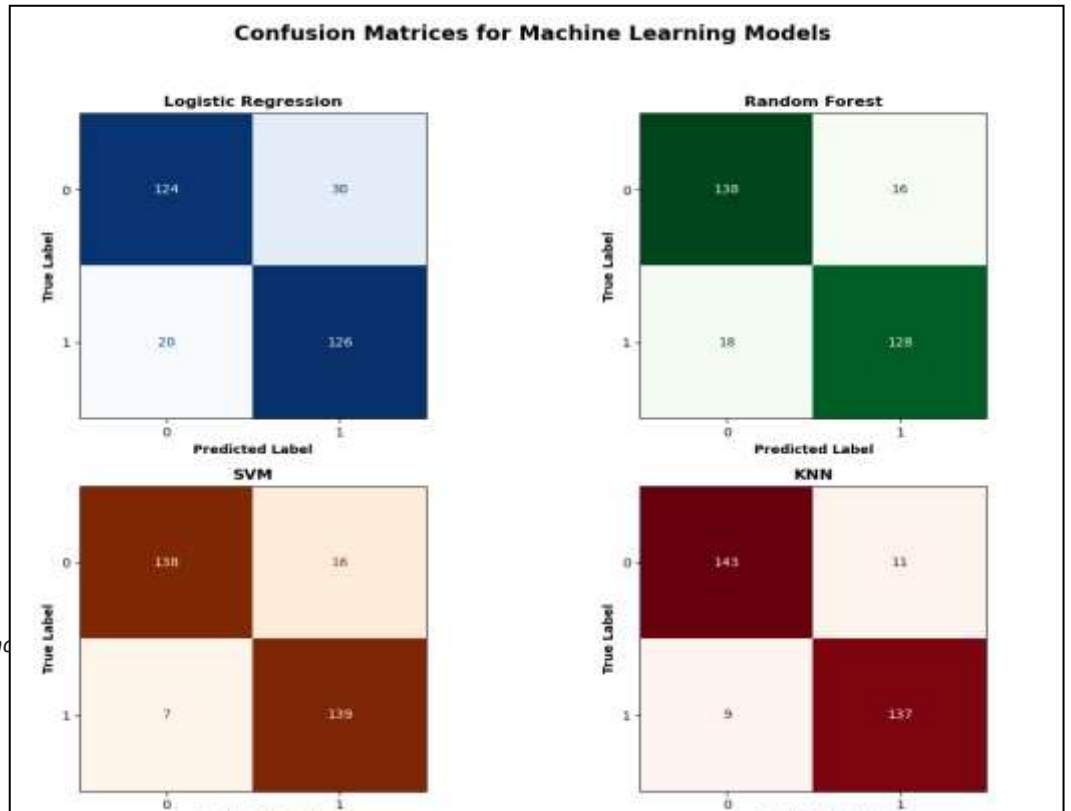
4.3 ROC Curves and Confusion Matrices

For a better understanding of model performance, ROC curves for each of the four classifiers were plotted. The highest AUC value of 0.982 was exhibited by the Random Forest model, which resulted in very good discriminatory potential between normal and anomalous sessions.

The SVM and LR curves were also very close to the top-left corner, validating their good performance.



Confusion and Random F arly important in anomaly detection scenarios. Failure to detect an anomaly might result in unnoticed system failure or breaches, thus models that aim to minimize such a mistake are ideal in real-world scenarios.



Nano

Figure 4: Confusion matrix for Machine learning models

5. Discussion

The results show that ensemble techniques such as Random Forest are very effective at identifying anomalies in software system logs because of their ability to manage feature interactions and avoid overfitting. The better performance of Random Forest confirms its appropriateness for structured log data where anomalies are context-dependent and sparse. The influence of feature representation stands out. Employing event count and TF-IDF vectorization made an important contribution to model performance since these methods both account for frequency and significance of events between sessions. More sophisticated representations like sequence modeling (e.g., LSTMs or Transformers) might still better capture temporal relationships and are an interesting direction for future research. The study has limitations despite high performance. The models were evaluated on a single dataset alone, although the HDFS logs are typical, and they do not reflect the entire range of diversity found in actual systems. Also, all models were run in an offline environment; real-time anomaly detection frameworks and integration of deep learning models for sequential modeling could be explored in future work.

6. Conclusion

In this study, the full scale evaluation of four classical machine learning algorithms for finding anomalies in software logs using the HDFS log datasets as the benchmark. The overall performance of Random Forest turned out the best, then Support Vector Machine, Logistic Regression and K-Nearest Neighbors. Random Forest outperforms because it is an ensemble based method which leverages the ensemble to effectively model non linear and reduce overfitting. Furthermore, by using consistent preprocessing techniques and feature extraction, the models could be compared on a fair and reliable basis. This work highlights the need of machine learning to complement the existing log monitoring system to provide more scalable and automated anomaly detection. It also contains actionable recommendations for choosing suitable ML models given tradeoffs in performance. We propose integrating deep learning methods, like LSTM and transformer based architectures and deploying these models into real time systems for proactive anomaly detection in dynamic software environments in the future.

References:

[1] Zhao, Z., Xu, C., & Li, B. (2021). A LSTM-based anomaly detection model for log analysis. Journal of Signal Processing Systems, 93(7), 745-751.
[2] Lee, Y., Kim, J., & Kang, P. (2023). Lanobert: System log anomaly detection based on bert masked language model. Applied Soft Computing, 146, 110689.
[3] Han, X., Yuan, S., & Trabelsi, M. (2023, December). Loggpt: Log anomaly detection via gpt. In 2023 IEEE International Conference on Big Data (BigData) (pp. 1117-1122). IEEE.
[4] Guo, H., Yuan, S., & Wu, X. (2021, July). Logbert: Log anomaly detection via bert. In 2021 international joint conference on neural networks (IJCNN) (pp. 1-8). IEEE.
[5] Alaca, Y., Basaran, E., & Çelik, Y. (2024). Enhancing Anomaly Detection in Large-Scale Log Data Using Machine Learning: A Comparative Study of SVM and KNN Algorithms with HDFS Dataset. ADBA Computer Science, 1(1), 14-18.
[6] Ali, S., Boufaied, C., Bianculli, D., Branco, P., & Briand, L. (2023). A Comprehensive Study of Machine Learning Techniques for Log-Based Anomaly Detection. arXiv preprint arXiv:2307.16714.
[7] Ryciak, P., Wasielewska, K., & Janicki, A. (2022). Anomaly detection in log files using selected natural language processing methods. Applied Sciences, 12(10), 5089.
[8] Duan, Y., Xue, K., Sun, H., Bao, H., Wei, Y., You, Z., ... & Ou, Z. (2024). LogEDL: Log Anomaly Detection via Evidential Deep Learning. Applied Sciences, 14(16), 7055.
[9] Liu, J., Huang, J., Huo, Y., Jiang, Z., Gu, J., Chen, Z., ... & Lyu, M. R. (2023). Log-based Anomaly Detection based on EVT Theory with feedback. arXiv preprint arXiv:2306.05032.
[10] Alaca, Y., Celık, Y., & Goel, S. (2023). Anomaly detection in cyber security with graph-based LSTM in log analysis. Chaos Theory and Applications, 5(3), 188-197.
[11] Le, V. H., & Zhang, H. (2022, May). Log-based anomaly detection with deep learning: How far are we?. In Proceedings of the 44th international conference on software engineering (pp. 1356-1367).
[12] Zhang, C., Peng, X., Sha, C., Zhang, K., Fu, Z., Wu, X., ... & Zhang, D. (2022, May). Deeptralog: Trace-log combined microservice anomaly detection through graph-based deep learning. In Proceedings of the 44th International Conference on Software Engineering (pp. 623-634).
[13] Chen, Z., Liu, J., Su, Y., Zhang, H., Ling, X., Yang, Y., & Lyu, M. R. (2022, May). Adaptive performance anomaly detection for online service systems via pattern sketching. In Proceedings of the 44th international conference on software engineering (pp. 61-72).
[14] Bovenzi, G., Aceto, G., Ciuonzo, D., Montieri, A., Persico, V., & Pescapé, A. (2023). Network anomaly detection methods in IoT environments via deep learning: A fair comparison of performance and robustness. Computers & Security, 128, 103167.
[15] Pranto, M. B., Ratul, M. H. A., Rahman, M. M., Diya, I. J., & Zahir, Z. B. (2022). Performance of machine learning techniques in anomaly detection with basic feature selection strategy-a network intrusion detection system. J. Adv. Inf. Technol, 13(1).
[16] Zipfel, J., Verworner, F., Fischer, M., Wieland, U., Kraus, M., & Zschech, P. (2023). Anomaly detection for industrial quality assurance: A comparative evaluation of unsupervised deep learning models. Computers & Industrial Engineering, 177, 109045.