AI-Based Error Detection in Web Applications Using Machine Learning Techniques

Ramneet Singh Chadha[1], Poonam Mishra[2]

[1]Scientist F, Embedded Systems, Center for Development of Advanced Computing (CDAC), Noida, Uttar Pradesh, India
[2]Project Engineer, Embedded Systems, Center for Development of Advanced Computing (CDAC), Noida, Uttar Pradesh, India

Abstract
Nowadays, web applications have become an essential part of business operations. They help employees complete tasks more quickly and efficiently, support better communication within organizations, and make the sharing and distribution of information easier and more effective. When APIs do not function properly, businesses can face serious consequences such as financial losses and reduced customer satisfaction. The traditional monitoring methods are usually able to detect the problems after the fact and thus lack of the ability to predict the behavior of complex error patterns. This research examines the application of machine learning for the detection of problems in web applications, utilizing the API Failure Intelligence Dataset (AFID) obtained from Kaggle. The data is first cleaned and refined to remove inconsistencies and improve its quality. In addition, domain knowledge is integrated into the dataset to make the model training process more meaningful and effective. Then, the performance of three machine learning models—Logistic Regression, Random Forest, and XGBoost—is evaluated to identify the root causes of API failures. The ensemble-based models performed well, achieving around 86% accuracy. However, a detailed analysis highlighted an inability to detect minority error classes, mainly due to the class imbalance in the dataset.

Keywords: API Error Detection, Machine Learning, Web Applications, AFID Dataset, Random Forest, XGBoost, Class Imbalance, Feature Engineering.

1. Introduction

APIs are an important part of modern web applications. They help applications connect to databases, manage user logins, and use external services [1]. However, when APIs face problems like invalid requests, slow responses, or services becoming unavailable, it can affect the application and sometimes even cause it to crash. These issues are usually detected using logging and manual monitoring. But they can be very slow, reactive and not enough for large application that receives thousands of API requests per minute [2]. A technique for identifying errors in application programming interfaces (APIs) is machine learning [3]. It does this by looking at historical logs and identifying patterns associated with different types of failures. Organizations can significantly reduce downtime by using automated methods to detect the root causes of issues. Therefore, this increases reliability.

The primary objective of this research is:

1. Preprocessing and feature engineering of API logs from the AFID dataset.
2. Adding information from experts to machine learning models to make them more accurate.

3. Evaluating the effectiveness of various supervised learning models in identifying root causes.
4. Using models to improve the functioning of web applications.

2. Literature Review

Error Detection in Web Applications: The subject has been thoroughly researched, with methods that vary from simple threshold monitoring to complex error prediction algorithms based on artificial intelligence. Early detection was usually done by going through log files manually or using simple rules, like sending an alert if the response time went above five seconds [2]. However, no method considers the complexity of the faults or changing demand, making it inefficient. Supervised learning techniques, like as boosting algorithms like XGBoost and tree-based ensembles like Random Forests, have demonstrated efficiency in identifying and forecasting system log outcomes [4,5].These models are robust even when the data is noisy or corrupted, can handle high-dimensional feature sets, and are capable of capturing non-linear relationships in the data. By managing the learning process, domain-specific knowledge added to machine learning pipelines increases model accuracy [6,7]. Heuristic rules, for example, are able to connect noisy labels to real failure patterns. This makes it possible for models to focus on significant connections. Even with these advancements, problems still remain, particularly when dealing with unbalanced datasets [8]. The limitations of the dataset make it hard to accurately detect rare errors and develop predictive models that work well in changing environments. This study contributes to the existing body of knowledge by combining expert views, utilizing ensemble learning strategies, and employing thorough feature preprocessing approaches. The primary objective of this research project is to enhance the detection of API problems in web applications.

3. Dataset Description

A significant collection of API failure logs is provided by the API Failure Intelligence Dataset (AFID) [9]. This dataset has than 2,20,000 entries. Each of these entries in the dataset is described by features. These features tell about the details of API requests and what happens as a result of these API requests.

3.1 Features

The main features of this dataset are:

- It has features like latency in milliseconds, request size in bytes, response size in bytes and retry count.
- Categorical: status_code and endpoint_name.
- Target: root_cause, which denotes the specific error type.

3.2 Data Characteristics

The dataset has several important characteristics:

1. The dataset is not balanced at the start. Some problems like database failures happen a lot often than other problems. This imbalance in the dataset really harm the models ability to find errors in the categories that do not happen often. The model has a time with the categories that are not common like the ones that are related to database failures because database failures are a big part of the dataset. The dataset imbalance is a problem for the model when it is trying to predict errors in the common categories, such, as the ones that are not related to database failures.
2. Secondly, the raw root cause distribution is noisy, presenting an almost random appearance and necessitating the incorporation of domain expertise.
3. Finally, there is a time dependency. Important information regarding the type of failures can be obtained by analyzing latency and retry patterns.

Table 1. Sample of the Dataset

latency_ms	request_size_bytes	response_size_bytes	retry_count	status_code	root_cause
120	1024	2048	0	200	CPU throttling
9500	512	1024	2	500	Database connection failure
300	40960	1024	1	400	Invalid request payload

The dataset is like what happens in the world when people use APIs. It includes several varying factors, such as response time, request size, and how the system behaves when errors occur and retries are needed. This makes the dataset really good for testing how well Artificial Intelligence(AI) can find errors. The dataset is good for this because it has a lot of variety, like the world with different latency, different request sizes and different retry behaviors.

4. Methodology

The proposed system's overall workflow is shown in Figure 1.

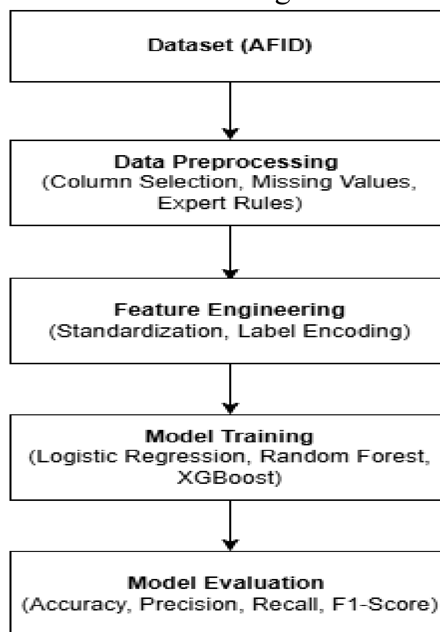

Figure 1. Proposed System Workflow for API Error Detection

4.1 Data Preprocessing

1. Column Selection: Columns that did not significantly improve the model's performance were removed. This included fields such as timestamp, service owner, and error message. These elements were not improving the model's performance in any meaningful way, therefore this was not a random process.
2. Handling Missing Values: Remove any rows with missing data to maintain the integrity of the model.
3. Expert Knowledge Injection: There was a few problems in the original dataset. About 85% of the data was improved using heuristic rules based on existing knowledge of a subject.

These rules were:

- latency_ms > 8000 → Database connection failure

- retry_count >= 3 → Downstream service unavailable
- status_code = 500 → Null pointer exception

The remaining 15% of the data was kept to preserve the natural variability, which ultimately helps the model's robustness. This method improves the model's learning by making the labels more consistent. At the time it keeps some of the original data so the model still gets some variation from the original data. Because it increases label consistency, which is essential for the model to learn, this approach is beneficial.

4.2 Feature Engineering

In this step the raw API data was prepared for the machine learning models. The numbers of things like latency_ms request_size_bytes and response_size_bytes, as well as Retry_count were made to be on the same level. This is done using the StandardScaler. The machine learning models can learn easily when latency_ms, request_size_bytes and response_size_bytes are on the same level. The categorical things, like status_code and endpoint_name were turned into numbers. Machine learning models cannot work with words so status_code and endpoint_name had to be turned into numbers. The target variable, which is the root_cause was turned into integer labels. This is because root_cause needs to support types of classification. This step of making the data better is important. It helps the model work properly. The data becomes neat and consistent. The API data and the root_cause are now ready for the machine learning models. The machine learning models can train on the API data and the root_cause. This helps the machine learning models work well. The machine learning models learn properly with algorithms. The machine learning models and the API data are ready, for the step.

4.3 Model Selection

We evaluated three supervised learning models:

Model	Description
Logistic Regression	Linear baseline model; interpretable but limited for complex relationships.
Random Forest	Ensemble of decision trees; handles non-linear data and reduces overfitting [4].
XGBoost	Gradient boosting algorithm; known for high predictive performance on tabular data [5].

Hyperparameters were set based on common best practices:

Model	Key Hyperparameters
Logistic Regression	max_iter=1000, random_state=42
Random Forest	n_estimators=100, max_depth=15, random_state=42
XGBoost	max_depth=10, eval_metric='mlogloss', random_state=42

5. Experimental Results

5.1 Model Evaluation Metrics

The comparison shows that ensemble learning methods work better than the basic Logistic Regression model when it comes to classifying API failure categories. This is because Logistic Regression has a lower accuracy and F1-score, which shows that it is not good at capturing complex non linear relationships in the dataset. This is to be expected because linear models don't work well with log data that is very different from each other and has a lot of dimensions. On the other hand, both Random Forest and XGBoost perform

much better because they can model non linear interactions and capture complex feature dependencies [4,5]. XGBoost has the highest accuracy and F1-score, which is a small improvement over Random Forest. This means that gradient boosting techniques perform better for optimizing and learning structured tabular data like API logs [5]. However, even though the models perform well overall, they struggle to correctly identify less frequent error classes. This is mainly due to class imbalance in the dataset [8]. As a result, the recall for rare failure categories is low, even when overall accuracy appears high. So, while ensemble models are effective for classification, additional techniques are needed to better detect the minority classes. These can include resampling methods to handle class imbalance.

Table 2. Model Performance Metrics

Model	Accuracy	Precision	Recall	F1-Score	Training Time
Logistic Regression	0.5894	0.4841	0.5894	0.5200	32.96 s
Random Forest	0.8570	0.7896	0.8570	0.8197	2.30 s
XGBoost	0.8582	0.7904	0.8582	0.8215	6.02 s

6. Discussion

The logistic regression and other linear models did not work well. This is because the connections between the input variables and the underlying causes were not easy to understand. They were complicated. When added some knowledge to the models by using established rules of thumb the models became better. This made the targets clear. The models then got an accuracy of about 86%. The logistic regression and other linear models worked better with this knowledge. The important things that helped the regression and other linear models were the numerical features. Specifically latency and retry_count were very important, for the regression and other linear models. These proved to be the best predictors, consistent with domain knowledge [6]. However, there are still some problems. The models are not able to detect errors correctly. The models demonstrate considerable overall accuracy, though this metric is biased due to the dominance of majority classes. The minority classes (rare API failures) are either misclassified or completely ignored (precision and recall scores for them are missing). This highlights the limitations of the current methodology in dealing with the imbalanced datasets [8]. Future research efforts could be directed towards addressing a class imbalance using techniques such as oversampling techniques (e.g., SMOTE [10]), anomaly detection approaches and hierarchical classification methods, with the aim of improving the prediction of rare error types. The results highlight the importance of combining domain knowledge and machine learning for identifying errors in real-world systems.

7. Conclusion

This study shows that machine learning techniques are really good at finding mistakes in web applications by looking at API log data. The models tested like Random Forest and XGBoost worked better than the Logistic Regression model. They were more accurate and better in showing things. The ensemble learning methods are good for log data that has a lot of details like seen in the results. This is because when domain knowledge added to the preprocessing step it makes the labels better and helps the models learn. This shows that there is need to use data analysis and expert knowledge to actual situations to make it work. The problem is that the data is not balanced and all models do not perform well in finding common types of errors. This limitation shows that accuracy alone is not sufficient to measure performance in these cases.

More evaluation methods are required. The proposed approach helps organizations find API failures early. This method also helps to reduce the time that systems are not working. It makes web applications more reliable. However, further improvements are still needed to better identify rare but critical errors.

8. Future Work

Future research should focus on improving how different types of errors are identified. This can be improved using techniques like Synthetic Minority Over-sampling Technique (SMOTE), cost-sensitive learning, and hybrid resampling methods. These methods help the model perform better in detecting API failures, especially the rare ones. In addition, models like Recurrent Neural Networks (RNNs) and transformer-based architectures can be used to capture patterns and relationships in API logs over time. Another key direction is developing systems that can detect errors in real time. This can be done by deploying trained models in production using streaming frameworks, enabling continuous monitoring and quicker responses to failures. This helps reduce system downtime.

Future work can also look at combining anomaly detection with machine learning techniques and using more diverse datasets to improve performance and make the system more reliable. Overall, this improves how well the model works in real-world web application environments.

References:

1. R.T. Fielding, Architectural Styles and the Design of Network-based Software Architectures, Doctoral dissertation, University of California, Irvine, 2000.
2. Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael I. Jordan. 2009. Detecting large-scale system problems by mining console logs. In Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles (SOSP '09). Association for Computing Machinery, New York, NY, USA, 117–132.
3. Varun Chandola, Arindam Banerjee, and Vipin Kumar. 2009. Anomaly detection: A survey. ACM Comput. Surv. 41, 3, Article 15 (July 2009), 58 pages.
4. Breiman, L. Random Forests. Machine Learning 45, 5–32 (2001).
5. Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16). Association for Computing Machinery, New York, NY, USA, 785–794.
6. E. Breck, S. Cai, E. Nielsen, M. Salib and D. Sculley, "The ML test score: A rubric for ML production readiness and technical debt reduction," 2017 IEEE International Conference on Big Data (Big Data), Boston, MA, USA, 2017, pp. 1123-1132.
7. Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin, “Why Should I Trust You? Explaining the Predictions of Any Classifier,” in Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), 2016, pp. 1135–1144.
8. H. He and E. A. Garcia, "Learning from Imbalanced Data," in IEEE Transactions on Knowledge and Data Engineering, vol. 21, no. 9, pp. 1263-1284, Sept. 2009.
9. M. Y. A. Baig, “API Failure Intelligence Dataset (AFID),” Kaggle, 2026.
10. N.V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, “SMOTE: Synthetic Minority Over-sampling Technique,” Journal of Artificial Intelligence Research, vol. 16, pp. 321–357, 2002.