AI-Driven Techniques for Noise Filtration in Software Validation Logs

Tarun Arora

tarun.arora@intel.com

Abstract

This paper presents a comprehensive examination of the enablement and integration of Natural Language Processing (NLP) and Random Forest Classifier (RFC) techniques to streamline log file diagnostics within a pivotal software development project. This novel integration has significantly improved the accuracy of classifying errors and informational queries in log files, marking a considerable advancement over traditional diagnostic methods. Our approach leverages advanced NLP to efficiently process and interpret the extensive, complex data within log files. In tandem with the robust classification capabilities of RFC, our method identifies and categorizes failure signatures with remarkable precision.

The effectiveness of our methodology and its potential to substantially reduce manual intervention in system diagnostics are underscored by these results. This innovation not only advances the field of software diagnostics into a new era characterized by automation and precision but also establishes a strong foundation for future technological progress. As technology evolves, the insights from this research have the potential to transform the maintenance and reliability of complex computing systems, indicating a significant paradigm shift in the practice of technological diagnostics.

Biography

Tarun Arora is working as senior engineer with Intel Technologies Limited, India. He is a seasoned change management and analytics professional with qualified rich experience working with global teams and fortune 500 companies across various domains. He led efforts to drive project management, quality management, customer delivery, business intelligence solutions and handcrafted process automation solutions. Successful projects with savings ranging in million dollars trusted the stakeholders in the technical skills and leadership qualities.

He is graduated in Industrial Production Engineering, with continued skills enhancement through various courses including Executive Program in Business Intelligence, Six sigma black belt, Master of Total Quality Management, Agile practitioner etc. supported the vision of delivering value through service.

1 Introduction

Platform development projects are validated for thousands of use cases by various system integration and validation techniques. Product validation is a continuous phenomenon with multiple test cycles executed over a period till the time Platform Validation (PV) milestone is granted. Each of the tests executed as part of these validation cycles generates validation logs which vary in types and sizes. Test types could be basic acceptance test, sanity test, stability, functional, Tape In, pre-integration and many others. These tests generate various log files, including .log, .csv, .err and each log has varied format. Even .log files generated for different tests have different formats. More importantly, these test cycles generate hundreds of logs weekly. Hence this is the classic case of Big Data 3 V - volume, velocity, and variety. Refer Figure1 (3V of Big Data).

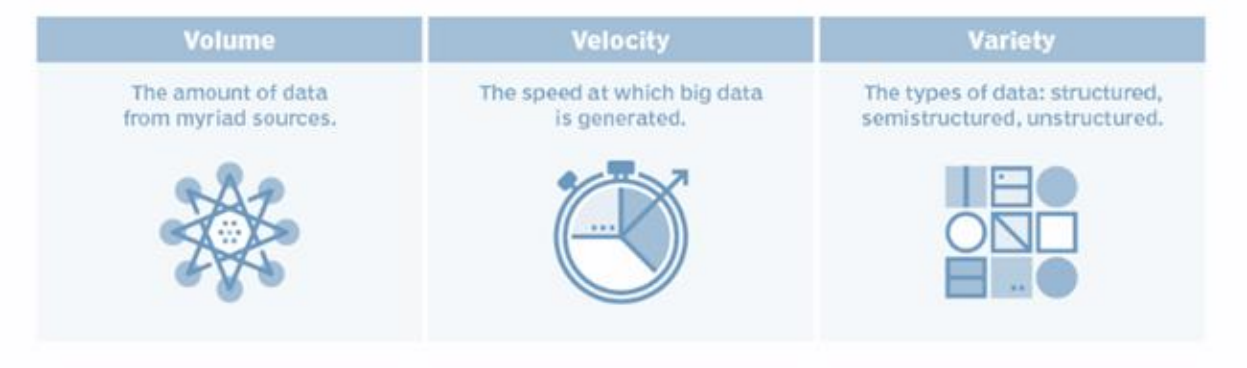

Figure1

2 Problem Description

Due to the 3V challenge of the validation logs data, the identification of real errors vs the noise elements in data is manual process and sometimes assisted with Regex operations. However, these Regex operations come with maintenance and become a nuisance for any new engineers in team. Also, there is always a possibility of missing errors due to human oversight. As reasoned above, overall efforts of filtering the noise from logs data and identifying real errors requires significant effort and time. This eventually leads to unproductive efforts and high triage or debug time which spans to ~1hrs to 2hrs per logs generated for a given test.

3 Solution

Quality and validation engineers worked together to develop a machine learning solution capable of predicting errors within validation logs and filtering out noise. The team identified that natural language processing (NLP) combined with classification modeling was the optimal approach. After creating and validating several models, the team successfully developed a proof of concept for the most effective machine learning model. This model automates the process of noise filtration and error triage in the logs. The image below illustrates two distinct workflows. The upper portion depicts the existing process, which involves a combination of manual effort and semi-automated tasks to identify error signatures. In contrast, the lower portion of the image represents the proposed integration of a machine learning model, designed to achieve a more efficient and effective end-to-end error detection system. Refer Figure2 below:

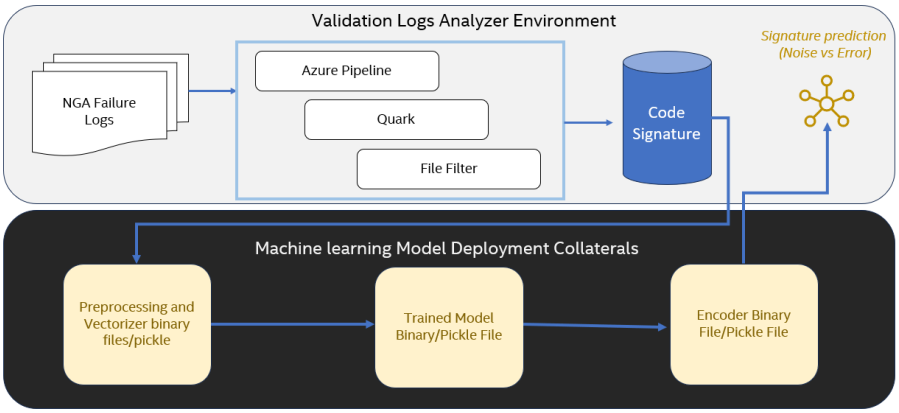

Figure2

4 Machine Learning: End to End Deployment Flow

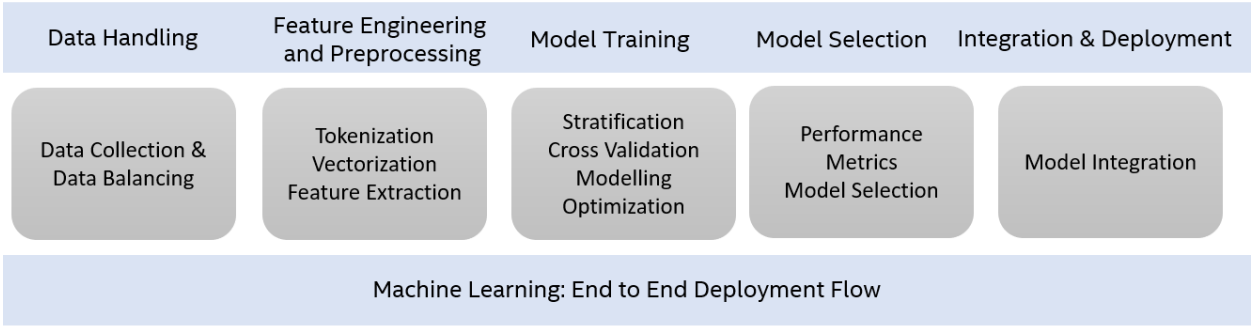

Figure3

5 Data Handling

The meticulous process of enhancing the triage tool for log file analysis started with a structured approach to data management. This process is pivotal for ensuring the foundation of our analysis is accurate, dependable, and tailored for optimizing diagnostics efficiency:

5.1 Data Collection

Logs dataset comprises essential elements such as log filenames, descriptors of queries, classifications of queries as errors or informational, alongside the substantive text of log signatures. “Table 1” provides a clear and organized overview of the dataset for easy reference and analysis.

File Name	Query Name	Query Type	Signature Text
XXX.log	ABC	error	XML oneof Check:

Table1

5.2 Guaranteeing Uniqueness of Data

To maintain the fidelity and integrity of our analysis, a crucial early step involves the removal of any replicative data entries. This ensures that each piece of data analyzed is unique, avoiding skewed results from duplicated information.

5.3 Streamlining Data Relevance:

The dataset undergoes a rigorous refinement process, where non-essential information is methodically stripped away. By isolating and removing columns that do not directly contribute to the failure diagnosis process, we focus our analysis on the most impactful and pertinent data elements.

5.4 Strategic Data Balancing:

Strategic Data Balancing plays a crucial role in addressing the inherent imbalance between 'error' and 'info' query types within our dataset.

Before Balancing Data:

INFO	ERROR
1488793	192565

To create a dataset conducive to unbiased machine learning model training, we implement a down sampling strategy. This involves equalizing the quantity of 'error' and 'info' categories by reducing the instances of the predominant class.
Mathematically, if the majority class has (Np) instances and the minority class has (Nl) instances, we randomly select (Nl) instances from the majority class to achieve a balanced dataset of (2Nl) instances. This harmonized class representation is essential for preventing model bias and ensuring that our analysis is both accurate and reflective of true log file dynamics, augmenting the triage tool's diagnostic capabilities.

After Balancing Data:

INFO	ERROR
192565	192565

Also, the StratifiedKFold method is employed with n_splits=5, ensuring that our data is divided into 5 distinct folds or splits for cross-validation, while maintaining an equal proportion of each class within every fold. StratifiedKFold, essential for handling imbalanced data, allows for a more reliable estimation of model performance by ensuring that each fold is a good representative of the overall class distribution. This technique is pivotal for reducing bias and variance in our model evaluation, supporting the authenticity of our performance metrics.
Stratified K-Fold cross-validation was used to validate the model's performance, defined by the formula:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Where (TP), (TN), (FP), and (FN) represent true positive, true negative, false positive, and false negative predictions, respectively.

6 Feature Engineering and preprocessing

Given the nature of log files, typically unstructured and cluttered with irrelevant information, preprocessing is a crucial step. The goal here is to convert raw text into a numerical format that machine learning models can understand. To achieve this, we first employ Regular Expressions (Regex) to meticulously extract significant patterns and entries from the log files. Regex allows us to sift through large datasets and pinpoint relevant information by matching specific patterns.

6.1 Our preprocessing pipeline involves the following key steps:

6.1.1 Tokenization and Stop Word Removal:

Tokenization involved breaking down the cleaned log entries into individual words or tokens. Following tokenization, we removed stop words - common words such as "the," "is," and "in," which typically do not contribute to the distinctiveness of log messages. This step was crucial for reducing the dimensionality of our data and improving the focus on keywords that could be indicative of system failures or errors. We utilize the get_stop_words('english') method to achieve this.

6.1.2 Lowercasing:

Converts all text to lowercase to ensure consistency, as machine learning models are case sensitive.

6.1.3 Lemmatization:

Each token then underwent lemmatization, a process of reducing words to their base or dictionary form. For instance, "running" becomes "run". We employed the getLemma function for this purpose, focusing on nouns as primary entities in log data. This standardization of vocabulary significantly contributes to the efficiency of the subsequent feature extraction phase, which involved transforming our processed textual data into a numerical format.
For a word (w) in our text (T), if (w) is not a stop word and its lemmatized form is (lemma(w)), our filtered text (T') can be represented as:

$$T' = \text{lemma}(w) | w \in T \ \wedge \ w \notin \text{stopwords}$$

6.2 Feature Extraction:

The TF-IDF (Term Frequency-Inverse Document Frequency) vectorizer was employed here, converting the corpus of preprocessed logs into a matrix of TF-IDF features. The TF-IDF metric represents the importance of a word within a document relative to a given corpus, balancing the frequency of the word in the document against its commonness across all documents. Mathematically, TF-IDF for a term (t) in a document (d) within a document set (D) is calculated as:

$$TF-IDF(t,d) = TF(t,d) \times IDF(t,D)$$

where (TF(t, d)) is the Term Frequency, representing the number of times term (t) appears in document (d)., and (IDF (t, D) is the Inverse Document Frequency, with (N) being the total number of documents, and (df(t)) denoting the number of documents containing term (t), computed as:

$$IDF(t, D) = \log \frac{N}{|d \in D : t \in d + 1|}$$

with (N) being the total number of documents in the corpus (D) and ($|d \in D : t \in d|$) representing the number of documents where the term (t) appears.
The logarithmic scaling of (IDF) diminishes the weight of terms that occur very frequently across the corpus, hence prioritizing unique terms in each document. Incorporating n-gram ranges (1,2) enables the vectorizer to not only consider individual terms (1-gram) but also pairs of consecutive terms (bi-grams), enriching the feature set and potentially capturing more contextual information useful for model training.

7 Model Training

7.1 Training and Testing by Data Splitting:

We precisely tailored our dataset for the Random Forest Classifier, implementing a stratified split to ensure a representative mix of 'error' and 'info' query types in both training (80%) and testing (20%) sets. This approach, facilitated by the train_test_split function with a set random_state, guards against bias and upholds the integrity of our log file classification model’s evaluation, significantly contributing to the enhancement of the triage tool's diagnostic accuracy and overall reliability. This meticulous preparation underpins our dedication to refining automated diagnostics.

7.2 Model Training Enhanced with Machine Learning:

Following the data preparation phase, we focus on the objective optimization and validation of our model's performance. This stage is characterized by a well-structured approach that leverages the RandomForestClassifier, integrated with GridSearchCV for hyperparameter tuning, and StratifiedKFold for robust cross-validation. This methodology is aimed at refining our predictive

model for enhanced accuracy in distinguishing between 'error' and 'info' query types in log files. Here is a breakdown of this meticulous process:

7.2.1 Random Forest Classifier:

A RandomForestClassifier instance (RF_GS_CV) is initialized as our base model. RFC is an ensemble learning method for classification that operates by constructing many decision trees at training time and outputting the mode of the classes (classification) of the individual trees. Random forests correct for decision trees' habit of overfitting to their training set, offering a more generalized model.

The decision tree, the building block of RFC, works by selecting the best feature among a random subset of features at each node to split the data. This randomness, coupled with bagging (Bootstrap Aggregating) - where multiple models are trained on different subsets of the training data and then averaged - ensures that the bias-variance trade-off is balanced. Mathematically, the Gini impurity, a measure used by the decision trees in the RFC to decide how to split the data, is given by:

$$G = 1 - \sum_{i=1}^{n} p_i^2$$

where (pi) is the frequency of class (i) in the dataset.

7.2.2 Hyperparameter Optimization with GridSearchCV:

To fine-tune our model, we employ GridSearchCV, applying it to the RandomForestClassifier with a set of predefined hyperparameters - 'n_estimators' (the number of trees in the forest) and 'max_depth' (the maximum depth of each tree). This exhaustive search over specified parameter values is conducted across k-fold cross-validation to ensure the model's generalizability. Mathematically, the optimization aims to maximize a performance metric (F), defined over the hyperparameter space (H), as:

$$\hat{h} = \arg\max_{h \in H}; F(h)$$

Where (ĥ) represents the optimal hyperparameter setting.
The performance of the model was evaluated based on accuracy, and the best parameters were chosen based on cross-validation scores, ensuring robust generalization to unseen data.

7.2.3 Model Training:

With the optimal hyperparameters determined, we train the RandomForestClassifier on the training subset transformed via TF-IDF vectorization, striving for a model that accurately discerns between 'error' and 'info' queries based on the textual content of log files.

8 Model Selection

The evaluation employed a suite of performance metrics—accuracy, precision, recall, and the F1-Score—each reflecting a distinct aspect of the model's classification capabilities:

- Accuracy: Serving as a primary measure, accuracy quantifies the overall correctness of the model's predictions across the test dataset, offering an aggregate perspective on its performance.
- Precision: This metric assesses the model's reliability in identifying true positive instances among all positive predictions, crucial for scenarios where the cost of false positives is high.

- Recall (Sensitivity): Recall measures the model's capacity to detect all actual positive instances, paramount in understanding its effectiveness in identifying relevant classifications without omission.
- F1-Score: The F1-Score amalgamates precision and recall into a single metric, providing a balanced view of the model's precision and recall capabilities by computing their harmonic mean.

We conducted a thorough evaluation of various diagnostic models, including logistic regression, naive Bayes classifier, support vector machine, random forest, and gradient boosting, and we also optimized their hyperparameters.
Based on the performance metrics, we narrowed our final selection to two candidates: Random Forest and Support Vector Machine.
Finally, we compared the performance of RandomForestClassifier (RFC) with that of Support Vector Machine (SVM) to identify the most effective tool for classifying 'error' and 'info' query types in log files. The analysis incorporated a detailed examination of predictive accuracy, particularly focusing on the incidence of false positives and false negatives, alongside metrics of precision and recall, to offer a granular understanding of each model's efficacy.

- SVM Performance Analysis:
  - The SVM model yielded a false positive and negative rate of approximately 0.5%, indicating a scenario where 0.5% of cases might be misclassified as 'info' when they are 'error', and vice versa. This rate, while low, underscores a critical challenge in distinguishing between the two query types with high reliability using SVM.
- RFC Performance Insights:
  - In contrast, the RFC model exhibited a significantly lower false-negative rate for 'error' predictions at 0.05%, albeit with a false positive rate for 'info' at around 3%. This suggests that while RFC is markedly adept at identifying 'error' queries with high coverage, it also presents a tendency to misclassify some 'info' queries as 'errors'.
- Precision and Recall Metrics:
  - Precision, evaluating the correctness of predictions for a given label, and recall, assessing the model's proficiency in accurately identifying correct labels out of the total population in the given data, served as pivotal metrics. The RFC model showed commendable precision in its predictions, coupled with a superior recall rate, highlighting its ability to effectively identify 'error' classifications from the dataset.

Following this comparative analysis, Figure3, highlights the performance metrics of various models evaluated during the process with focus on SVM and RFC models, providing a visual summary of their predictive efficacy, error handling, and overall capacity to accurately classify log file queries.

Metrics	TFIDF_SVM	TFIDF_Logit	TFIDF_NB	TFIDF_RS_GS_CV Est: 50 , Depth: None	TFIDF_GB Est: 100 , Depth: 10, learningrate: 0.5	TFIDF_SVM	TFIDF_RS_GS_CV Est: 50 , Depth: None
FN	368	399	816	363	366	346	169
TP	60988	60949	59318	60997	60991	60993	263069
FP	263	302	1933	254	260	258	1484
TN	60860	60829	60412	60865	60857	60882	36946
FPR	0.60%	0.65%	1.36%	0.59%	0.60%	0.56%	3.86%
TPR	99.40%	99.35%	98.64%	99.41%	99.40%	99.44%	99.94%
TNR	99.57%	99.51%	96.90%	99.58%	99.57%	99.58%	96.14%
FNR	0.43%	0.49%	3.10%	0.42%	0.43%	0.42%	0.064%

Table2

To complement these quantitative assessments, a Confusion Matrix was constructed, offering a visually detailed account of the model's predictive outcomes. The confusion matrix delineates the distribution of true positives, true negatives, false positives, and false negatives, providing nuanced insight into the model's classification behavior. This visualization aids in identifying patterns or biases in the predictions, further contextualizing the numerical metrics. Following this, Figures 4 and 5 are presented, highlighting the confusion matrices for RFC and SVM models, respectively. These matrices serve as crucial tools for visually interpreting the performance of each model in classifying 'error' and 'info' queries in log files, allowing for a deeper understanding of their predictive accuracy and areas for improvement.

	error	info
error	36946	169
info	1484	263069

Table3: Random Forest Confusion Matrix with Stratification Strategy

	error	info
error	60882	346
info	258	60933

Table4: SVM Confusion Matrix with Down sampling Strategy

This array of performance metrics, coupled with the confusion matrix, furnishes a comprehensive understanding of the RFC model's predictive prowess.

The comparative analysis of RFC with traditional Support Vector Machine (SVM) models demonstrated the former's superior performance in minimizing false-negative rates and enhancing precision in error query identification. This underscores our methodology's effectiveness and its potential to significantly reduce manual intervention in system diagnostics.

9 Integration and Deployment

The integration phase fuses the trained RandomForestClassifier (RFC) model with Triage tool, granting it the capability to autonomously identify and categorize failure signatures in log files through advanced NLP techniques and RFC’s classification power. This harmonization significantly boosts Triage tool's automated log analysis efficiency, aligning with project goals to elevate system diagnostics' precision and reliability.

- The selection and implementation of the RFC-based model in Triage tool highlighted substantial improvements in the tool's performance. One of the major outcomes was the ability to create more accurate and granular failure signatures without explicit pre-programming. This advancement meant a decrease in the need for manual interventions during the failure analysis phase. Additionally, the RFC model improved the predictive accuracy of Triage tool, allowing it to adapt to new and unseen failure modes.
- Implemented Standalone Model and infra for testing Model in Millions of files from every project.
- Analyzed log in Host Patch Update logs and identified many occurrences of Patch upload failure.

By presenting these results, we contribute novel insights into automatic log file analysis, underscoring the viability of employing advanced machine learning algorithms and NLP techniques for refined diagnostic endeavors. The clarity and precision of these results underscore our commitment to advancing the field and offer a significant contribution to the technological community's understanding of automated system diagnostics.

After rigorous training and meticulous hyperparameter tuning, the embedded Random Forest Classifier (RFC) model has enhanced Triage tool's ability to process raw textual data. It now efficiently extracts actionable insights, driving maintenance strategies to new levels of effectiveness.
This pivotal step underscores our commitment to leveraging innovative technology for enhancing software diagnostics and ensures a broad impact on system reliability and maintenance methodologies across the technological landscape. The figure below illustrates the integration of the machine learning model with the validation log analyzer within the final environment.

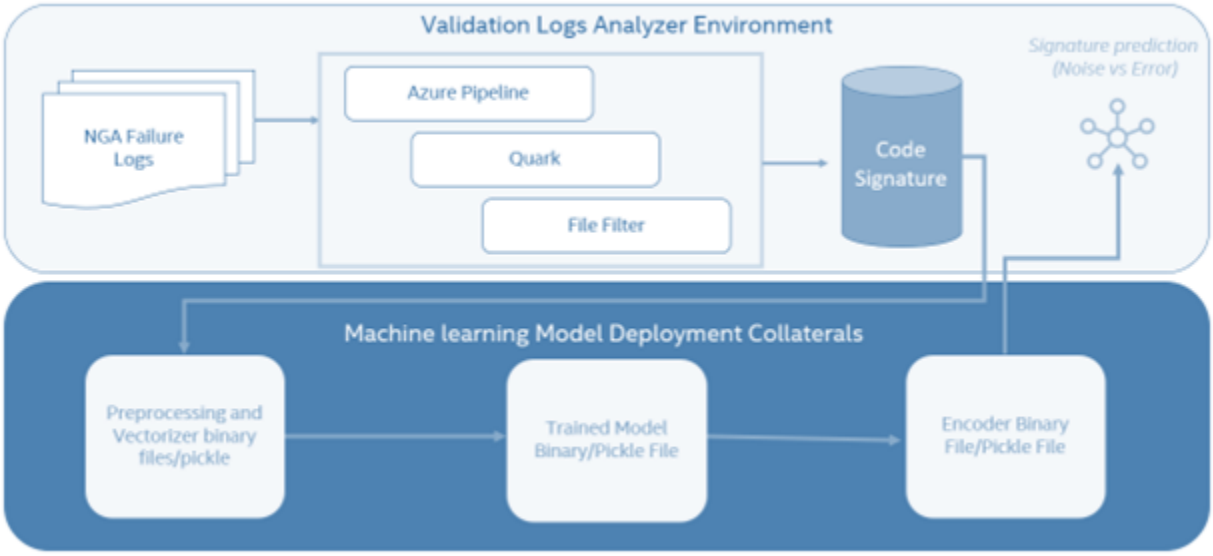

Figure4

10 Conclusion

We embarked on a comprehensive assessment of integrating Natural Language Processing (NLP) and RandomForestClassifier (RFC) techniques for streamlining log file diagnostics. This integration, the first of its kind, notably enhanced the accuracy of classifying errors and information queries in log files, signifying a substantial step forward from traditional diagnostics methods. Our approach utilized advanced NLP to efficiently process and interpret the vast and complex data contained within log files, coupled with the classification power of RFC, to identify and categorize failure signatures with unprecedented accuracy and efficiency.

As outlined in the problem description section, the analysis of each log file generated by a given test typically requires 1 to 2 person-hours, depending on the size of the log. With an average of 10,000 log files being generated weekly, we can conservatively estimate a potential time saving of up to 10,000 to 20,000 person-hours per week. This estimate assumes that the implementation of our solution would eliminate the need for manual analysis, thereby saving a significant amount of effort.
In conclusion, this machine learning project asserts the significant potential of marrying NLP with RFC within the domain of log file analysis. It not only propels the field of software diagnostics into a new era marked by automation and precision but also sets a robust foundation for future advancements. As technology continues to advance, the insights from this study bear the potential to revolutionize the maintenance and reliability of complex computing systems, signaling a radical shift in how technological diagnostics are approached and implemented.