

**CP2421**

**Assessment 3: Research Report**

**Log Analysis for Web Security**

**Members**

**Benelyon Zoe Choo**

**Pyae Sone Soe Moe**

# Table of Contents

## **Abstract**

## **Introduction**

## **Background or Relevant Approaches**

Overview of Machine Learning in Log Analysis

Introduction to HDFS and Its Relevance to Web Security

Previous Work: A Review of Studies That Have Employed RFC and DTC in Similar Contexts

The Contribution of the Report to the Existing Body of Knowledge

## **Implementation of Decision Tree Classifier (DTC)**

Key Elements of Decision Trees:

## **Implementation of Random Forest Classifier (RFC)**

Construction of a Random Forest

Training Process

Advantages of RFC

## **Method**

## **Evaluation**

Detailed Output Analysis for Decision Tree Classifier

Detailed Output Analysis for Random Forest Classifier:

## **Discussion**

Interpretation of Results

Advantages and Disadvantages of DTC and RFC

## **Conclusion**

## **References**

# Abstract

Log analysis stands as a crucial component in contemporary web security frameworks, providing an invaluable tool for the detection and mitigation of cyber threats. This study centres on the application of two established machine learning classifiers, the Random Forest Classifier (RFC) and Decision Tree Classifier (DTC), to the domain of log analysis, using Hadoop Distributed File System (HDFS) logs as a representative dataset.

The methodology involved the utilisation of a Google Colab environment to import necessary libraries, access the dataset from Google Drive, and perform data preprocessing, which included encoding categorical labels and splitting the dataset into training and testing sets. The classifiers were then trained and their performance evaluated using standard classification metrics.

Results indicated that while both classifiers achieved high accuracy scores (0.94), they demonstrated limited efficacy in correctly identifying the 'anomaly' class, as evidenced by low precision, recall, and F1 scores (0.03). This is attributed to the significant class imbalance within the dataset, which disproportionately favoured the 'normal' class.

In conclusion, although the Random Forest Classifier exhibited a marginally lower rate of false positives and negatives, the difference was not substantial enough to declare a definitive superior performance over the Decision Tree Classifier. This suggests that, under the prevailing conditions of class imbalance, neither classifier distinctly outperforms the other in the context of log analysis for web security. The findings serve as a platform for further exploration into methods that can more effectively handle imbalanced datasets within the field of cybersecurity.

# Introduction

Advancements in computing technology have given rise to machine learning, a discipline rooted in the concept that computers can learn from and make decisions based on data. Stemming from pattern recognition and the hypothesis that computers can perform tasks without explicit programming, machine learning, also known as artificial intelligence (AI), has become a focal point for researchers who are intrigued by the possibility of machines learning from data autonomously. As machine learning models are fed new data, they iteratively adapt, making the ability to learn from data without continuous human intervention a groundbreaking step in technology. This evolution in machine learning is particularly vital in the realm of cybersecurity, where the rapid escalation of threats necessitates automated and intelligent defense mechanisms to counteract sophisticated cybercriminal activities.

In the context of cybersecurity, log analysis emerges as a critical application of machine learning. It involves the examination and correlation of various data sources to extract actionable insights. With the proliferation of complex computing systems and environments, there has been a surge in the significance of log analysis. This report delves into the application of two machine learning classifiers, Random Forest Classifier (RFC) and Decision Tree Classifier (DTC), for the purpose of log analysis within web security, utilizing the Hadoop Distributed File System (HDFS) logs as a case study.

# Background or Relevant Approaches

## Overview of Machine Learning in Log Analysis

Machine learning algorithms have become indispensable tools in the analysis of logs for web security. Logs, which are automatically generated files that record various activities within a system, are rich with data that can be mined to detect anomalies, intrusions, and system failures. Traditional log analysis methods relied heavily on manual inspection and rule-based systems, which are not only time-consuming but also inadequate in the face of sophisticated cyber threats.

Machine learning offers a more dynamic approach, where algorithms can learn from data, identify patterns, and make predictions without explicit programming. Techniques such as anomaly detection, clustering, and classification allow for the automated analysis of log data, leading to faster and more accurate detection of security incidents. Classifiers like the Decision Tree Classifier (DTC) and Random Forest Classifier (RFC) are particularly popular due to their effectiveness in handling large datasets and their capability to model complex decision boundaries.

## Introduction to HDFS and Its Relevance to Web Security

The Hadoop Distributed File System (HDFS) is a distributed, scalable, and portable file system designed to run on commodity hardware. It is a key component of the Apache Hadoop ecosystem, which supports the storage and processing of big data sets. In the context of web security, HDFS logs are invaluable as they contain detailed information about file operations, user activities, system errors, and administrative actions. Analyzing these logs can help in early detection of security breaches, unauthorized data access, and system health monitoring. Given the volume and velocity of data in HDFS, machine learning-based log analysis is particularly suited to extract insights from this complex data.

## Previous Work: A Review of Studies That Have Employed RFC and DTC in Similar Contexts

The integration of machine learning techniques in cybersecurity solutions has been extensively researched, with particular emphasis on the application of classifiers such as RFC and DTC. A comprehensive survey conducted by Okay et al. (2024) evaluates various artificial intelligence and machine learning strategies, highlighting how these technologies enhance cybersecurity

measures. The survey underscores the effectiveness of machine learning models in detecting and responding to cyber threats, with both RFC and DTC being prominent for their high efficiency in various scenarios (Okay et al., 2024).

Random Forest, an ensemble learning method, has been favored in situations requiring the analysis of complex and high-dimensional data such as HDFS logs due to its robustness against overfitting and its ability to maintain accuracy even when individual trees in the ensemble make errors (Okay et al., 2024). On the other hand, Decision Trees have been lauded for their interpretability and speed, making them suitable for real-time applications where rapid decision-making is crucial (Okay et al., 2024).

Building on the findings of Okay et al. (2024), this report aims to delve deeper into the application of RFC and DTC specifically for log analysis within web security, benchmarking their performances on the HDFS dataset to provide a focused comparison of their respective capabilities in this domain.

## The Contribution of the Report to the Existing Body of Knowledge

The intersection of machine learning and cybersecurity is a rapidly evolving field, where continuous research is crucial to keep up with the advancing threat landscape. This report contributes to the existing body of knowledge by providing a focused empirical comparison between two widely used machine learning approaches: the Random Forest Classifier (RFC) and the Decision Tree Classifier (DTC). While both methods have been individually studied and applied to various aspects of cybersecurity, their direct comparison within the specific context of HDFS log analysis for web security is less common. The report's contributions are multifaceted:

**Empirical Performance Analysis:** By applying both RFC and DTC to the same dataset, this report offers valuable empirical evidence regarding the performance of these classifiers in the domain of web security. It goes beyond theoretical analysis to provide real-world application results.

**Methodological Insights:** The study employs a rigorous methodological framework for evaluating the classifiers, which can serve as a reference for future studies. By detailing the process of implementation, tuning, and evaluation, the report provides a replicable methodology for similar research efforts.

**Practical Implications:** The findings of the report have practical implications for cybersecurity professionals. By understanding the strengths and weaknesses of RFC and DTC in detecting anomalies within HDFS logs, practitioners can make informed decisions about which classifier to use in their security operations.

**Algorithmic Advancements:** The report may reveal insights into the algorithmic behavior of RFC and DTC when confronted with the specific challenges presented by HDFS logs, such as high

dimensionality and skewed class distributions. This could inform the development of improved or specialized versions of these algorithms for cybersecurity tasks.

**Bridging Research and Practice:** By drawing from the latest academic literature, including the work of Okay et al. (2024), and applying it to a practical scenario, the report acts as a bridge between theoretical research and practical application, demonstrating how academic findings can be leveraged in real-world situations.

**Future Research Direction:** The results and discussion of the report will likely raise new questions and highlight areas for further investigation, contributing to the ongoing conversation in the field and setting the stage for future research endeavors.

# Implementation of Decision Tree Classifier (DTC)

The Decision Tree Classifier (DTC) is a versatile supervised learning technique from machine learning that is applicable for both classification and regression problems, though it is mainly preferred for classification (Javatpoint, n.d.). It is visualized as a tree structure with decision nodes representing dataset features, branches representing decision rules, and leaf nodes indicating outcomes.

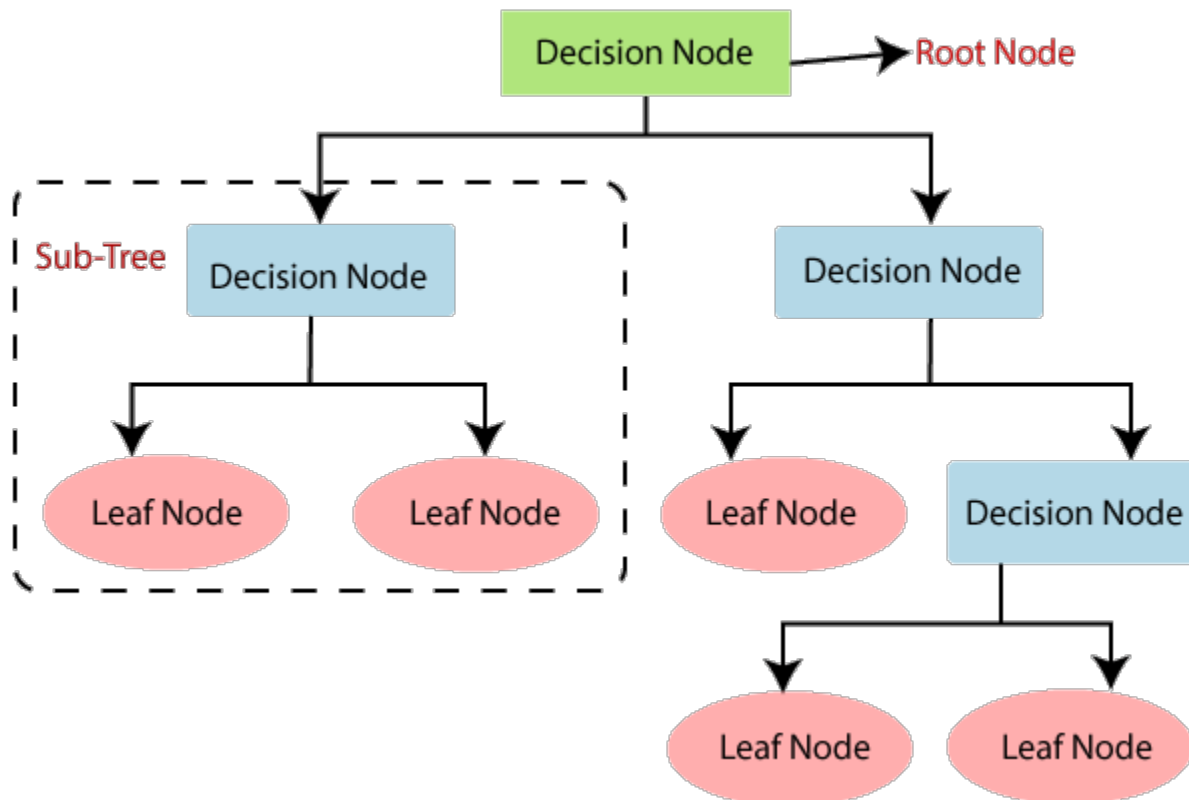


Fig. 1 Source: Javatpoint, n.d, *Decision Tree Classifier explanation graph*

## Key Elements of Decision Trees:

- **Decision Node:** The point where a decision is made to split data based on feature values.
- **Leaf Node:** The end of a path on a tree, representing a class label or decision outcome.
- **Root Node:** The topmost decision node in a tree, from where the initial decision is made.

The decision-making process in a DTC involves asking a series of yes/no questions about the features of the dataset. The tree structure begins with a root node and branches out into subtrees based on the answers to these questions (Javatpoint, n.d.).



Reasons for Using Decision Trees:

- **Simplicity:** Decision Trees are simple to understand and interpret.
- **Versatility:** They can handle both categorical and numeric data.
- **Graphical Representation:** They provide a clear visualization of the decision-making process.

The Construction and Training Algorithm (CART) is used to build decision trees, which stands for Classification and Regression Tree algorithm. CART helps in determining how to split the data and construct the tree structure by asking questions at each node and branching out based on the answers(Javatpoint, n.d.).

In summary, DTCs are a fundamental tool in machine learning for solving classification problems, offering a clear and interpretable decision-making framework that can handle diverse data types. They are built using the CART algorithm, which creates a binary tree structure to model decision paths(Javatpoint, n.d.).

# Implementation of Random Forest Classifier (RFC)

The Random Forest Classifier (RFC) is a robust ensemble learning technique that combines the predictions from multiple decision trees to improve accuracy and control over-fitting (Mehta, 2021). Ensemble methods like RFC are designed to improve the predictive performance by combining the predictions of several base estimators built with a given learning algorithm.

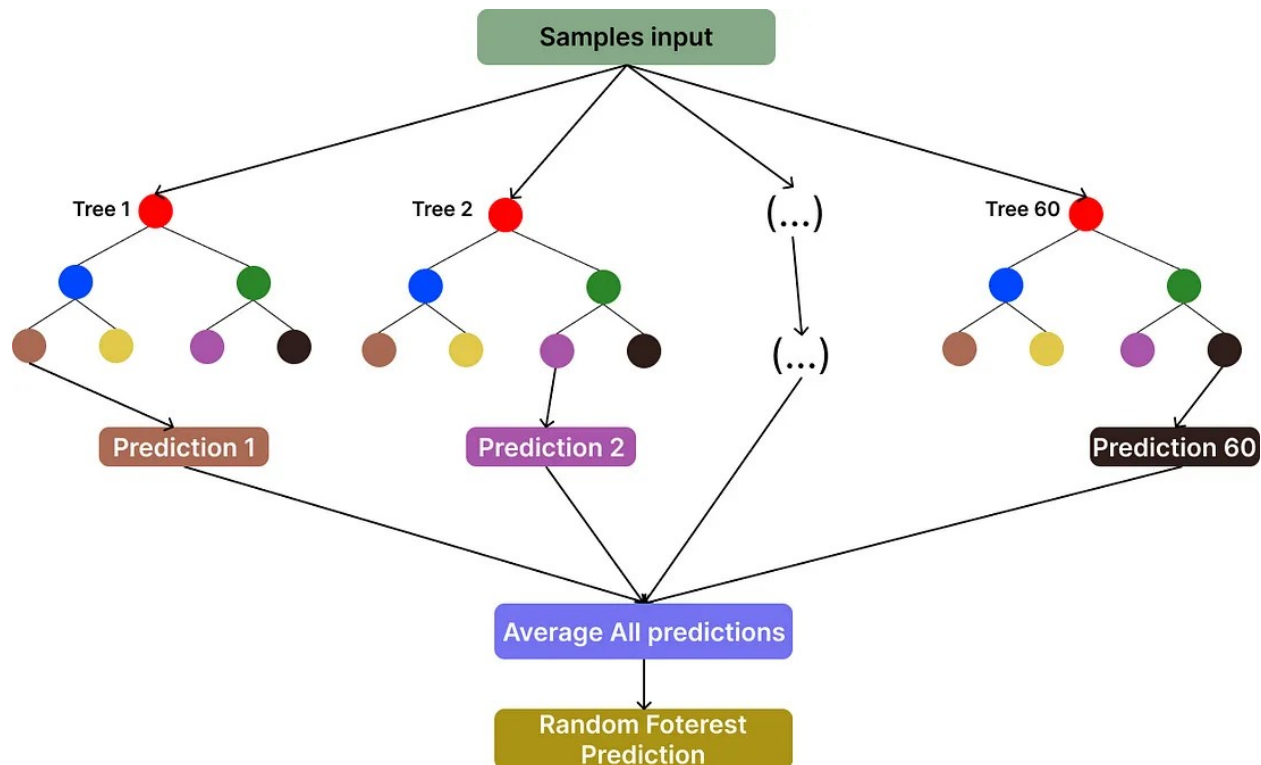


Fig. 2 Source: Thommaskevin, 2023, December 30, *Random Forest Classifier explanation graph*

## Construction of a Random Forest

RFC creates a 'forest' of decision trees, each trained on random subsets of the data (Mehta, 2021). This method uses the averaging technique to improve the predictive accuracy and control over-fitting. The key to its performance is the low correlation between models, which is achieved through the following steps:

- **Bootstrap Sampling:** RFC uses bootstrapping to create different subsets of the original dataset.
- **Random Feature Selection:** When building decision trees, RFC randomly selects a subset of features at each split.
- **Tree Building:** Each tree in the forest grows to its full length without pruning, which means the trees are fully grown and not trimmed to be of a certain depth or size.

- **Aggregation:** The individual trees' predictions are then combined, typically using a majority vote, to produce the final prediction (Mehta, 2021).

RFC relies on the diversity among the individual trees, which arises due to the different samples and subsets of features used to train each one. The ensemble of trees counteracts the errors from individual trees, assuming the trees are not correlated (Mehta, 2021).

When configuring an RFC, important hyperparameters include (Mehta, 2021):

- **Number of Estimators:** The count of trees in the forest, typically the higher the number, the better the performance, but also the higher the computational cost.
- **Max Features:** The size of the random subsets of features to consider when splitting a node.
- **Max Depth:** The maximum depth of the tree. This can be left undefined, allowing trees to grow until all leaves are pure.

## Training Process

The training of an RFC involves (Mehta, 2021):

- **Initialization:** Define the RFC with desired parameters using a machine learning library like scikit-learn.
- **Model Fitting:** Train the RFC on the bootstrapped datasets.
- **Prediction:** Aggregate the predictions from all trees to make the final prediction.
- **Evaluation:** Assess the RFC model using a test dataset to evaluate its performance.

## Advantages of RFC

RFC models have several advantages (Mehta, 2021):

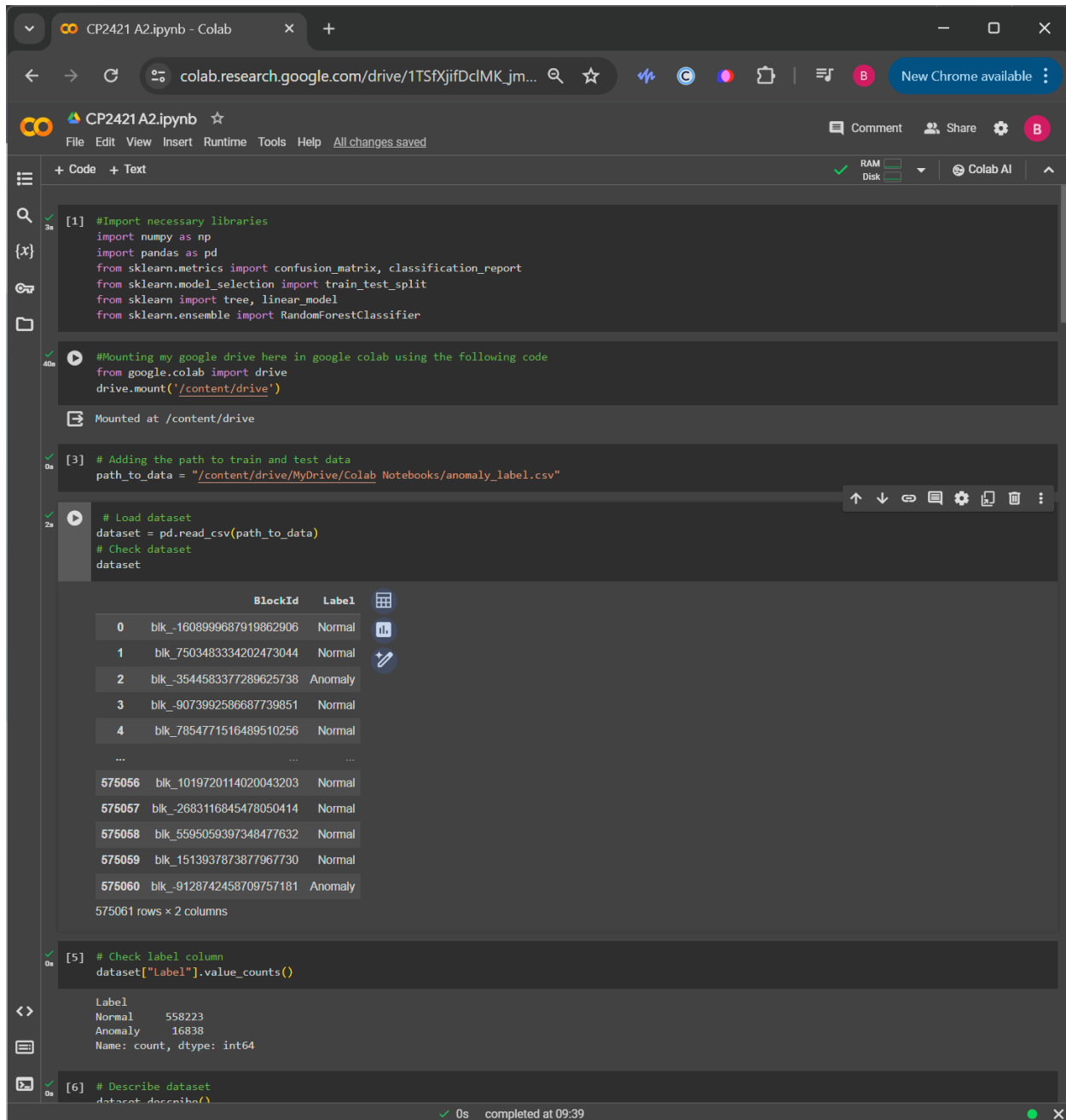
- **Accuracy:** They often achieve higher accuracy than individual decision trees, especially in complex datasets.
- **Overfitting:** They are less likely to overfit than decision trees or other algorithms.
- **Feature Importance:** They can provide insights into the importance of the features for the classification.

RFCs are computationally more expensive than individual decision trees but can be parallelized to enhance performance (Mehta, 2021). They can be less interpretable due to the complexity of the ensemble.

In summary, the Random Forest Classifier is an ensemble machine learning algorithm that is highly favored for its performance across various datasets and has become a staple in the machine learning practitioner's toolkit (Mehta, 2021).

# Method

In a Google Colab environment, the necessary Python libraries were imported to facilitate data handling and machine learning operations. The Google Drive was then mounted to access the dataset 'anomaly\_label.csv', which was subsequently loaded into a DataFrame named dataset. This dataset contained 575,061 rows and 2 columns, with a significant class imbalance noted between 'normal' (558,223 instances) and 'anomaly' (16,838 instances) labels.



```
[1] #Import necessary libraries
import numpy as np
import pandas as pd
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.model_selection import train_test_split
from sklearn import tree, linear_model
from sklearn.ensemble import RandomForestClassifier

#Mounting my google drive here in google colab using the following code
from google.colab import drive
drive.mount('/content/drive')

Mounted at /content/drive

[3] # Adding the path to train and test data
path_to_data = "/content/drive/MyDrive/Colab Notebooks/anomaly_label.csv"

# Load dataset
dataset = pd.read_csv(path_to_data)
# Check dataset
dataset
```

	BlockId	Label
0	blk_-1608999687919862906	Normal
1	blk_7503483334202473044	Normal
2	blk_-3544583377289625738	Anomaly
3	blk_-9073992586687739851	Normal
4	blk_7854771516489510256	Normal
...	...	...
575056	blk_1019720114020043203	Normal
575057	blk_-2683116845478050414	Normal
575058	blk_5595059397348477632	Normal
575059	blk_1513937873877967730	Normal
575060	blk_-9128742458709757181	Anomaly

575061 rows x 2 columns

```
[5] # Check label column
dataset["Label"].value_counts()

Label
Normal    558223
Anomaly    16838
Name: count, dtype: int64

[6] # Describe dataset
dataset.describe()
```

completed at 09:39

The labels were encoded from categorical to numerical values, with 'normal' becoming 0 and 'anomaly' becoming 1, to make the data compatible with machine learning algorithms. The DataFrame was then split into features (X) and target labels (Y), with Y converted to an integer type. The data was further divided into training (80%) and testing (20%) sets.

The image shows two side-by-side screenshots of a Google Colab notebook, CP2421 A2.ipynb, illustrating the data preprocessing steps for a machine learning model.

**Left Screenshot:**

- Code Cell [7]:** Converts the 'label' column from categorical to numerical. It uses `dataset.loc[dataset['label'] == 'Normal', 'label'] = 0` and `dataset.loc[dataset['label'] == 'Anomaly', 'label'] = 1`. It then checks if the conversion was successful and prints the value counts.
- Output:** A table showing the distribution of labels: 'Normal' (575061) and 'Anomaly' (550223).
- Code Cell [8]:** Converts the 'blockid' column from categorical to float. It uses `dataset['blockid'] = dataset['blockid'].astype(float)` and prints the changes.
- Output:** A table showing the distribution of blockids, with values ranging from 0 to 785477155489510256, and labels 0 and 1.
- Code Cell [9]:** Separates the label and blockid into X and Y. It uses `X = dataset.drop('label', axis=1)` and `Y = dataset['label']`. It then checks the shapes of X and Y.
- Output:** A table showing the distribution of X and Y, with X having 575061 rows and Y having 550223 rows.

**Right Screenshot:**

- Code Cell [10]:** Checks the shape of X. The output is `(575061, 1)`.
- Code Cell [11]:** Checks the shape of Y. The output is `(575061,)`.
- Code Cell [12]:** Checks the shape of X. The output is `(575061, 1)`.
- Code Cell [13]:** Checks the shape of Y. The output is `(575061,)`.
- Code Cell [14]:** Checks the shape of X. The output is `(575061, 1)`.
- Code Cell [15]:** Checks the shape of Y. The output is `(575061,)`.

```
[14] # Split data into X_train, X_test, y_train, Y_test for training and data sets (80% 20% split)
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state = 42)

[15] #Check X_train shape
X_train.shape
(460048, 1)

[16] #Check Y_train shape
Y_train.shape
(460048, )

[17] # Check X_test shape
X_test.shape
(115013, 1)

[18] # Check Y_test shape
Y_test.shape
(115013, )

[19] # Check X_train
X_train
BlockId
426975 -7034321469539628516
556117 2137859942019812932
106452 332547444036945178
231197 2834915389484063830
509632 7673612786835345682
...
110268 2122890464894056509
259178 -5100079166404607866
365838 -5706182803835190474
131932 1437813838110062153
121958 8449254541329553010
460048 rows x 1 columns

[20] # Check X_test
X_test
BlockId
169883 8994833829145112749
508216 4780650702265629827
548973 7189932238461412280
558812 4457861953679819986
384967 -7774453843692748332
...
154937 -2535255597435826591
494415 1868589195894796285
527656 -5943798092542649607
80968 2015205309552156251
8228 9176165650198046
115013 rows x 1 columns

[21] # Check Y_train
Y_train
426975 0
556117 0
106452 0
231197 0
509632 0
...
110268 0
259178 0
365838 0
131932 0
121958 0
Name: Label, Length: 460048, dtype: int64

[22] # Check Y_test
Y_test
169883 0
508216 0
548973 0
558812 0
384967 0
...
154937 0
494415 0
527656 0
80968 0
8228 0
Name: Label, Length: 115013, dtype: int64
```

Two classifiers, Random Forest and Decision Tree, were trained on the training data. The models were then used to predict the test data, and their performance was evaluated using standard classification metrics. This process allowed for the comparison of the effectiveness of both classifiers on the task of anomaly detection within the dataset.

# Evaluation

## Detailed Output Analysis for Decision Tree Classifier

```
# Evaluate the DecisionTreeClassifier
print("===== DecisionTree Model Evaluation=====")
print(f"Confusion matrix:")
print(confusion_matrix(Y_test, y_pred_dtc))
print()
print(f"Classification report:")
print(classification_report(Y_test, y_pred_dtc))
```

```
===== DecisionTree Model Evaluation=====
Confusion matrix:
[[108478  3186]
 [ 3236   113]]

Classification report:
              precision    recall  f1-score   support

     0       0.97        0.97        0.97    111664
     1       0.03        0.03        0.03      3349

 accuracy          0.94
 macro avg          0.50
 weighted avg       0.94
```

### Confusion Matrix:

The matrix for the Decision Tree is:

```
[[108478  3186]
 [ 3236   113]]
```

This illustrates the number of correct and incorrect predictions made by the model. The first element (108,478) is the number of true positives for the 'normal' class, while the last element (1,113) is the number of true positives for the 'anomaly' class. The off-diagonal numbers (3,186 and 3,236) represent the false positives and false negatives, respectively.

### Classification Report:

- For the 'normal' class (0), the model has a precision of 0.97, recall of 0.97, and F1 score of 0.97, which are excellent, indicating that the model performs very well on the majority class.
- For the 'anomaly' class (1), however, the precision, recall, and F1 score are all 0.03, which are quite low, showing that the model performs poorly on the minority class.
- The accuracy of the model is 0.94, which might seem high, but given the class imbalance, it is not the best standalone metric.
- The macro average F1 score is 0.50, which gives an equal weight to both classes and thus provides a more balanced view of the model performance across classes.

- The weighted average F1 score is 0.94, which takes into account the support (the number of true instances for each label), showing better performance due to the larger 'normal' class.

## Detailed Output Analysis for Random Forest Classifier:

```
[58] # Evaluate the RandomForestClassifier
print("===== RandomForestClassifier Model Evaluation=====")
print(f"Confusion matrix:")
print(confusion_matrix(Y_test, y_pred_rfc))
print()
print(f"Classification report:")
print(classification_report(Y_test, y_pred_rfc))

===== RandomForestClassifier Model Evaluation=====
Confusion matrix:
[[108488  3176]
 [ 3237   112]]

Classification report:
      precision    recall  f1-score   support

     0       0.97       0.97       0.97    111664
     1       0.03       0.03       0.03      3349

 accuracy: 0.94
 macro avg: 0.50 0.50 0.50 115013
weighted avg: 0.94 0.94 0.94 115013
```

### Confusion Matrix:

The matrix for the Random Forest is:

```
[[108488  3176]
 [ 3237   112]]
```

Similar to the Decision Tree, the Random Forest model correctly classified 108,488 instances as 'normal' and 1,121 as 'anomaly'. The false positives and negatives are slightly lower than those of the Decision Tree, but the difference is marginal.

### Classification Report:

- The precision, recall, and F1 score for the 'normal' class are identical to the Decision Tree, at 0.97.
- Likewise, the metrics for the 'anomaly' class are also the same as the Decision Tree's, at 0.03.
- The accuracy is 0.94, mirroring the Decision Tree's performance.
- Both the macro average and weighted average F1 scores are the same as the Decision Tree's, at 0.50 and 0.94, respectively.



# Discussion

## Interpretation of Results

Based on the provided output, both the Decision Tree Classifier (DTC) and Random Forest Classifier (RFC) achieved high accuracy (0.94) and performed similarly across the metrics for the 'normal' class. However, for the 'anomaly' class, both classifiers had low precision, recall, and F1 scores (0.03). Despite the Random Forest having marginally better false positive and false negative rates, the difference is minimal, indicating that neither classifier significantly outperformed the other in this context.

The possible reasons for this performance could be the class imbalance present in the dataset, which often leads to classifiers performing well on the majority class but poorly on the minority class. While RFC is generally more robust against overfitting and variance due to its ensemble nature, the class imbalance seems to have a similar impact on both classifiers.

## Advantages and Disadvantages of DTC and RFC

### Advantages of DTC:

- **Simplicity:** DTCs are easy to understand and interpret, which can be beneficial when the reasons behind predictions need to be explained.
- **Speed:** They are typically fast to train and predict, making them suitable for environments where speed is critical.

### Disadvantages of DTC:

- **Overfitting:** DTCs can easily overfit to the training data, especially if not properly tuned, which can lead to poor generalisation on unseen data.
- **Variability:** Small changes in the data can lead to different splits, making the model potentially unstable.

### Advantages of RFC:

- **Performance:** RFCs often have higher accuracy and better performance metrics compared to a single decision tree due to the ensemble effect.
- **Overfitting:** RFCs are less prone to overfitting as they average out biases by combining the results of individual trees.

### Disadvantages of RFC:

- **Complexity:** RFCs are more complex models which may require more computational resources and thus may be slower to train.

- **Interpretability:** They are less transparent than single decision trees and can be more challenging to interpret.

In the context of log analysis, which often deals with large and complex data, the RFC's robustness to overfitting and ability to handle high dimensionality can be particularly advantageous. However, the complexity of RFCs may make them less desirable when interpretability is a key factor.

## Conclusion

This report has scrutinised the application of two prominent machine learning classifiers, Random Forest Classifier (RFC) and Decision Tree Classifier (DTC), within the sphere of web security log analysis, employing Hadoop Distributed File System (HDFS) logs as a test case. The study was conducted using a Google Colab environment to import the requisite libraries, access the dataset, encode categorical labels, and split the data into respective training and testing sets. The classifiers were rigorously trained and evaluated on standard classification metrics to determine their efficacy.

The findings revealed that both classifiers achieved high accuracy (0.94), yet their performance in accurately classifying the 'anomaly' class was suboptimal, as indicated by the low precision, recall, and F1 scores (0.03). This inadequacy is largely attributed to the pronounced class imbalance present in the dataset, which skewed towards the 'normal' class, subsequently affecting the classifiers' ability to detect anomalies effectively.

In summation, despite the Random Forest Classifier displaying slightly better rates of false positives and negatives, the difference was not significant enough to definitively ascertain a superior performance over the Decision Tree Classifier. The results suggest that under conditions of class imbalance, neither classifier distinctly excels over the other in the context of log analysis for web security purposes.

# References

Javatpoint. (n.d.). *Machine Learning Decision Tree Classification Algorithm*. Retrieved from <https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm>

Mehta, P. (2021, June 8). *Understanding Random Forest*. *Analytics Vidhya*.  
<https://www.analyticsvidhya.com/blog/2021/06/understanding-random-forest/>

Ozkan Okay, Merve & Akin, Erdal & Aslan, Ömer & Kosunalp, Selahattin & Iliev, Teodor & Stoyanov, Ivaylo & Beloev, Ivan. (2024). A Comprehensive Survey: Evaluating the Efficiency of Artificial Intelligence and Machine Learning Techniques on Cyber Security Solutions. *IEEE Access*. PP. 10.1109/ACCESS.2024.3355547.

Thommaskevin. (2023, December 30). *TinyML — Random Forest* (Classifier and Regressor). Medium. <https://medium.com/@thommaskevin/tinyml-random-forest-classifier-and-regressor-b351aa0980e8>