EV Charging Station Attack Detection Using Machine Learning

by

Kamran Athar Janwiri

B.Sc., University of Sindh, Pakistan, 2014

A Project Submitted in Partial Fulfilment of the Requirements for the Degree of

MASTER OF ENGINEERING

in the Department of Electrical and Computer Engineering

EV Charging Station Attack Detection Using Machine Learning

by

Kamran Athar Janwiri

B.Sc., University of Sindh, Pakistan, 2014

Supervisory Committee

Dr. Fayez Gebali, Supervisor

Department of Electrical and Computer Engineering

Dr. Mohamed W. El-kharashi, Co-supervisor

Department of Electrical and Computer Engineering

Abstract

Electric Vehicle (EV) charging stations are very important for supporting the adoption of EVs, but they are at risk of cyberattacks. This project looks at how Machine Learning (ML) can help to detect these attacks using the CICEVSE2024 dataset, which has data about normal operations and attack scenarios. ML models like Random Forest (RF), Gradient Boosting Machine (GBM), Support Vector Machines (SVM), Logistic Regression (LR), Naive Bayes (NB), and k-Nearest Neighbors (KNN) were tested. Principal Component Analysis (PCA) applied to simplify the dataset by minimizing the features, and SMOTE (Synthetic Minority Oversampling Technique) was used to balance the dataset. Models were evaluated with 21, 15, 10, and 5 features to find the best accuracy and speed. RF shows the best accuracy whereas KNN was the fastest.

Table of Contents

List of Figures

List of Tables

Abbreviations

ARFF…… Attribute Relation File Format

CSV…….. Comma-Separated Values

CIC…….Canadian Institute for Cybersecurity

CLI…… Command Line Interface

DOS…….. Denial of Service

EV…… Electric Vehicle

EVSE…… Electric Vehicle Supply Equipment

FPR……..... False Positive Rate

GBM…… Gradient Boosting Machines

GUI…….. Graphic User Interface

HPC……..... Hardware Performance Counters

KNN……..... K-Nearest Neighbors

LR……. Logistic Regression

ML……... Machine Learning

NB…… Naive Bayes

OCPP……... Open Charge Point Protocol

PCA……. Principal Component Analysis

RF……..... Random Forest

SVM…… Support Vector Machines

SMOTE…… Synthetic Minority Oversampling Technique

TPR……. True Positive Rate

V2G……. Vehicle-to-Grid

WEKA……. Waikato Environment for Knowledge Analysis

Acknowledgment

I thank Almighty Allah for His endless blessings throughout this journey. I am deeply grateful to my family, parents, wife, daughter, and son for their unwavering love and support.

Specifically, I thank my supervisor, Dr. Fayez Gebali, for his guidance and encouragement. I would also like to thank the University of Victoria for providing an excellent learning venue. I want to thank my brother Adnan Athar Janwari for always supporting me, I am also grateful to Abdul Aleem Syed, Tanveer Ahmed, and Muhammad Awais for their motivation and invaluable advice.

Dedication

This work is dedicated to my parents, wife, daughter, son, and brother Adnan Athar Janwari.

Chapter: 1 Introduction

The use of EVs is increasing as they are better for reducing the pollution and protecting the environment [1]. Countries like Canada has plans to spend $1.7 billion to make it affordable and easier for people to buy it. The government also has a plan to ensure that 100% of new light vehicles sold are electric by 2035 [2]. To support such transition of EVs, more charging stations, also called Electric Vehicle Supply Equipment (EVSE), are being built. These EV charging stations are important, but they are also at risk of cyber attacks. Such attacks can disrupt the service or compromise sensitive data, which makes security a big concern [3], [4].

This project uses the CICEVSE2024 dataset [4] to explore and look at how ML can help to detect cyber attacks on EVSE systems. First, SMOTE was used to ensure that the dataset is balanced [5], [6], furthermore, PCA was used to decrease the features [7], [8] with different sets of, such as 21,15,10 and 5, after evaluating all different sets of features. It was identified the top 10 features are the key features that contribute more to classifying the attack with the best accuracy and execution time. In this project, different ML models were evaluated to see how well they could classify the attack. The models accuracy and execution time were assessed using methods like 10-fold cross-validation and a 70/30% train-test split.

The results show that ML can enhance the EV charging station security. This project helps to find the best models and important features to detect attacks that help to improve the security of charging stations which allow people to switch to EVs, without worry and ensure safe and smooth operation.

1.1 Motivation

The motivation behind this project is to protect EVSE systems from these threats which can allow people to switch smoothly without being worried about any security threat. ML is a powerful tool for analyzing patterns in data which can help to detect cyber attacks [7], [9]. So, objective of this project is to find best ways to improve the security and reliability of EVSE systems using ML, which can ensure a smooth shift toward EVs.

1.2 Related Work

In this literature [4], CIC researchers have tried various cyberattacks, such as TCP Port Scan, Service Version Detection, OS Fingerprinting, Aggressive Scan, SYN Stealth Scan, Vulnerability Scan, Slowloris Scan, UDP Flood, ICMP Flood, PSHACK Flood, ICMP Fragmentation, TCP Flood, SYN Flood, SynonymousIP Flood, Cryptojacking, Backdoor, and out of these many of attacks were successful, the most impactful were Reconnaissance Technique, SYN Flood Attack, Backdoor Exploitation, Cryptojacking. At last, they did the machine learning experiment where they applied ML models such as RF, KNN, and SVM for binary and multi-class classification. These models were tested using techniques like PCA and 10-fold cross-validation [4].

To continue further on the previous study, this project focuses on binary classification, which simplifies the task by distinguishing only between attack and benign scenarios. So before moving forward first ensure that the dataset is clean for the models and to do so first dataset is cleaned to remove errors, duplicates, and irrelevant data. After cleaning the dataset, the different sets of features (21, 15, 10, and 5) were selected using PCA [7], [8], and six ML models, RF, GBM, SVM, KNN, LR, and NB were compared for their effectiveness in detecting attacks. These models were evaluated using 10-fold cross-validation and 70/30% train-test split, to ensure that the model is robust and reliable for the results [12], [13], and SMOTE was used to fix the imbalance data in the dataset [5]. Among all the models, RF achieved the best result with an accuracy of 95.9%, clearly showing its potential as an effective solution for detecting cyber attacks. This project advances the previous work by combining rigorous preprocessing steps, advanced feature selection, balancing the dataset, and comprehensive model evaluation, providing a practical and efficient approach to improving the security of EVSEs.

1.5 Report Organization

The report structure is outlined below.

Chapter 1: Introduction

This chapter gives the quick overview of the project, The related work and motivation for this project were also described.

Chapter 2: ML Models and Tools

This chapter introduced the ML models used to predict attacks on EVSE and explained the Weka tool used to implement them.

Chapter 3: Proposed Framework

This chapter proposed a prediction model for detecting attacks on EVSE, covering the testing setup, data preprocessing, feature selection, dataset balancing and models.

Chapter 4: Evaluation and Results

This chapter explained the metrics used to evaluate model performance and analyzed the results of different ML models, comparing their effectiveness in detecting cyber-attacks.

Chapter 5: Conclusion and Future Work

This chapter summarized the main findings and conclusions of study and suggested future research to improve EV charging station security.

Chapter 2: Brief Overview of Machine Learning (ML)

ML helps computers learn from historical data and improve at tasks as they receive more data without needing to be programmed for every specific situation. To decide if an activity is an attack or normal, models are trained on data and can make decisions automatically.

For EV charging stations (EVSE), ML can be absolutely useful. It can study things like power usage, network traffic, and system activities to learn what normal looks like. Once trained, it can rapidly identify abnormal patterns that might indicate that an attack is underway. For instance, in the case of DoS attack or crypto jacking, if there is a change in behavior, ML would observe the same and report it. This makes EV charging stations more secure, protecting these stations from hackers [7], [9].

2.1 Supervised Learning

Supervised Learning is a category of ML where the input examples are labeled. This means that the data has distinct classes such as attack or normal, which gives the computer a target to be modified. This set of labeled data is then used to train the algorithm. The computer sees many examples of both attack activity and normal activity. It then uses these examples to learn the patterns of the categories and to determine the category for new data. During training, a feedback mechanism is also in use to increase accuracy. When the computer makes a mistake, it is corrected and adjusts its learning. After the model is trained, it can learn on new data and classify it as attack or normal [7].

2.2 Classification

It is one of the most prevalent and successful forms of supervised learning. The algorithm is actually being taught through this process how to classify data into predefined classes like attack or benign. This model, after learning from the labeled dataset, can then study new data and classify it into one of the classes. Applications like, spam detection to fraud detection, while this project puts it to work in identifying cyber-attacks on EV charging stations to enhance their security [7], [14].

2.3 k-Fold Cross-Validation

Figure 2.1 shows how 5-fold cross-validation works in a simple way. In this example, the dataset is split into five equal parts, called "folds." Each fold is used as the test set once, while the other four folds are combined to train the model. This process is repeated five times, so every fold gets a chance to be the test set. After each round, the model's performance is calculated. At the end, the results from all five rounds are averaged to find the final accuracy. This method ensures the model is tested on all parts of the data, giving a fair and reliable evaluation of its accuracy and execution time [12], [13].

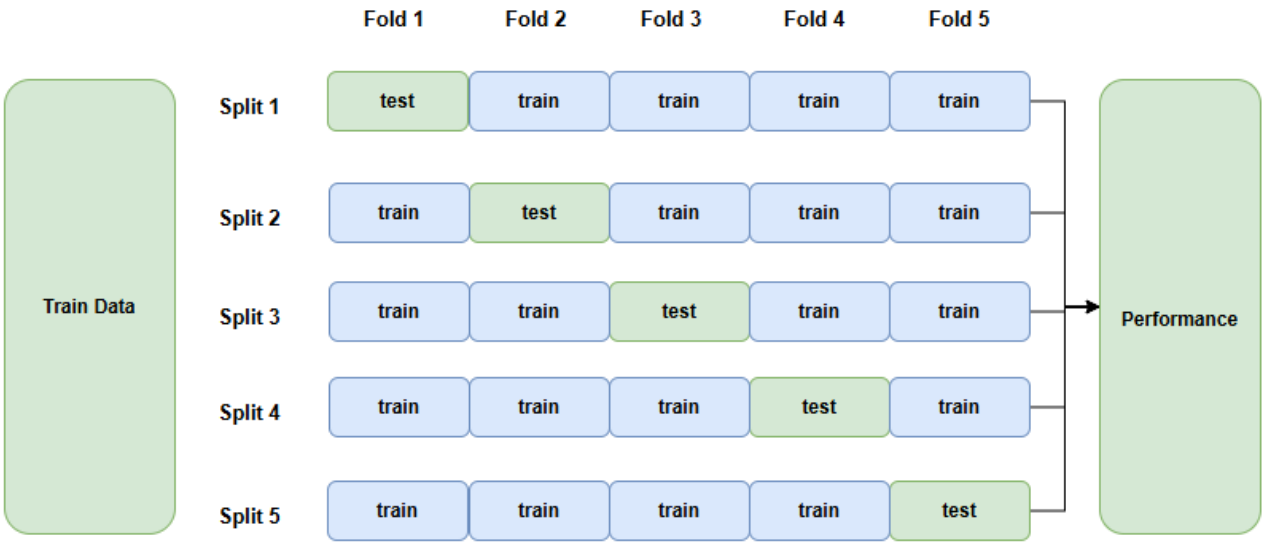


Figure 2.1 Visualization of 5-Fold Cross-Validation.

2.4 Overview of WEKA Interface

Figure 2.2 shows WEKA GUI, which is straightforward and user-friendly. WEKA is ML tool developed by the University of Waikato in New Zealand. It is available for free under the GNU General Public License. WEKA helps with various tasks such as data preparation, classification, regression, clustering, and data visualization. It is developed in Java technology, and it simplifies working with data and applying ML methods, making it an excellent tool for both students and professionals [15].

Figure 2.2 Overview of WEKA Interface.

WEKA can be used as CLI or GUI. As WEKA has quite simple and easy GUI, so we prefer to continue using the WEKA GUI in this project. Once click on the explore button it shows six main tabs, Preprocess, Classify, Cluster, Associate, Select Attributes, and Visualize. The Preprocess and Classify sections are the most used features in WEKA. The preprocess tab mainly being used to apply the preprocessing filters on the dataset like removing duplicate entries, adjusting data to a standard scale, reducing the number of samples, manage missing data. And Classify tab is when we want to run the model on the dataset [15], [16].

Figure 2.3 shows that WEKA can work with many different file types, including .csv and .arff, which are the most common. But the main format WEKA uses is .arff.

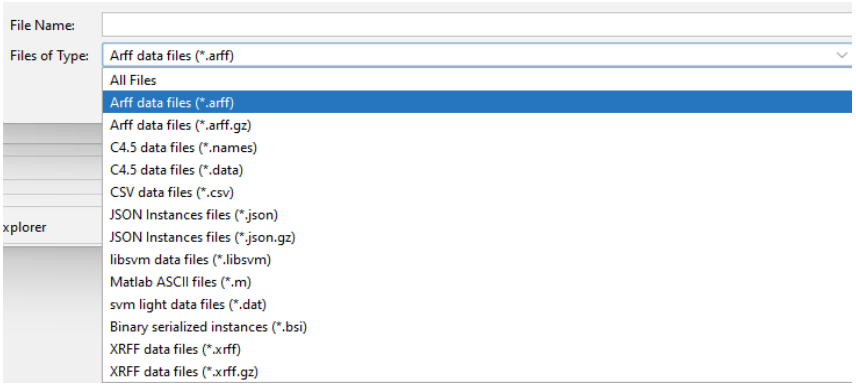


Figure 2.3 WEKA Preprocess Panel Overview.

Figure 2.4 shows the Preprocess panel in WEKA. This panel provides details about the dataset, such as the number of attributes (features) and instances (rows). It also displays information about each feature, including its type, range of values, missing data, and the class balance of the dataset.

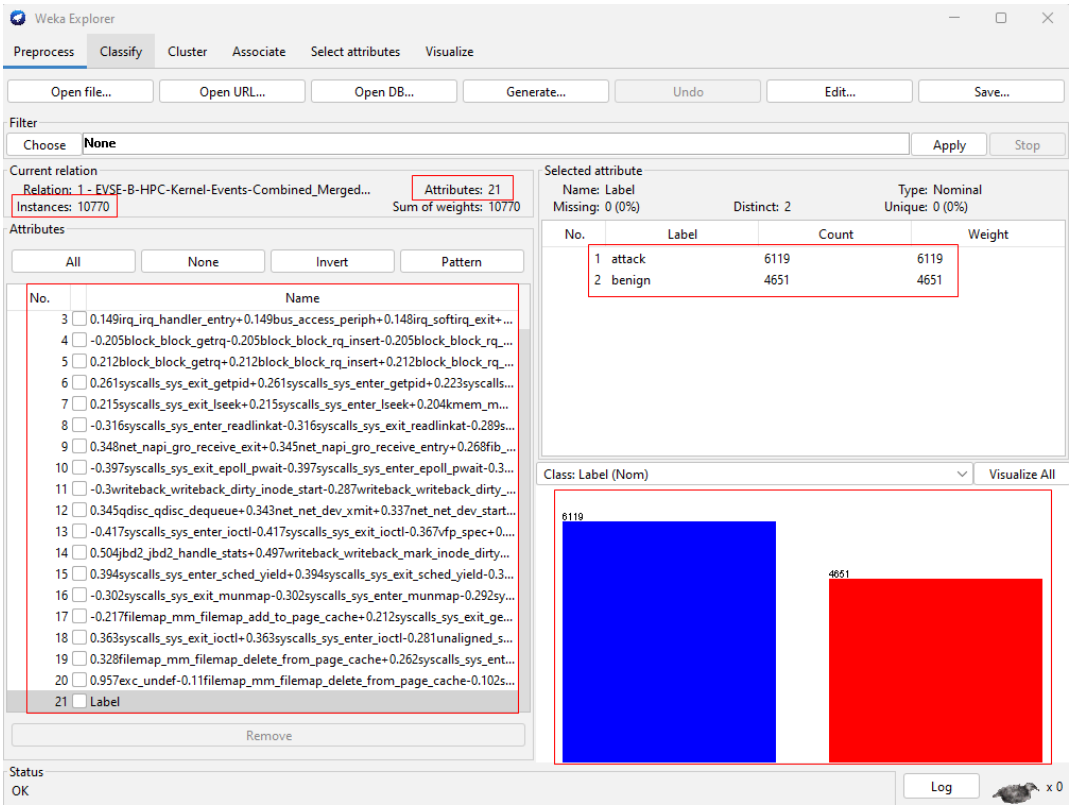


Figure 2.4 Overview of dataset.

Chapter 3: EV Charging Station Attack Detection

Figure 3.1 illustrates the architecture of the proposed framework for the EV charging station attack detection system.

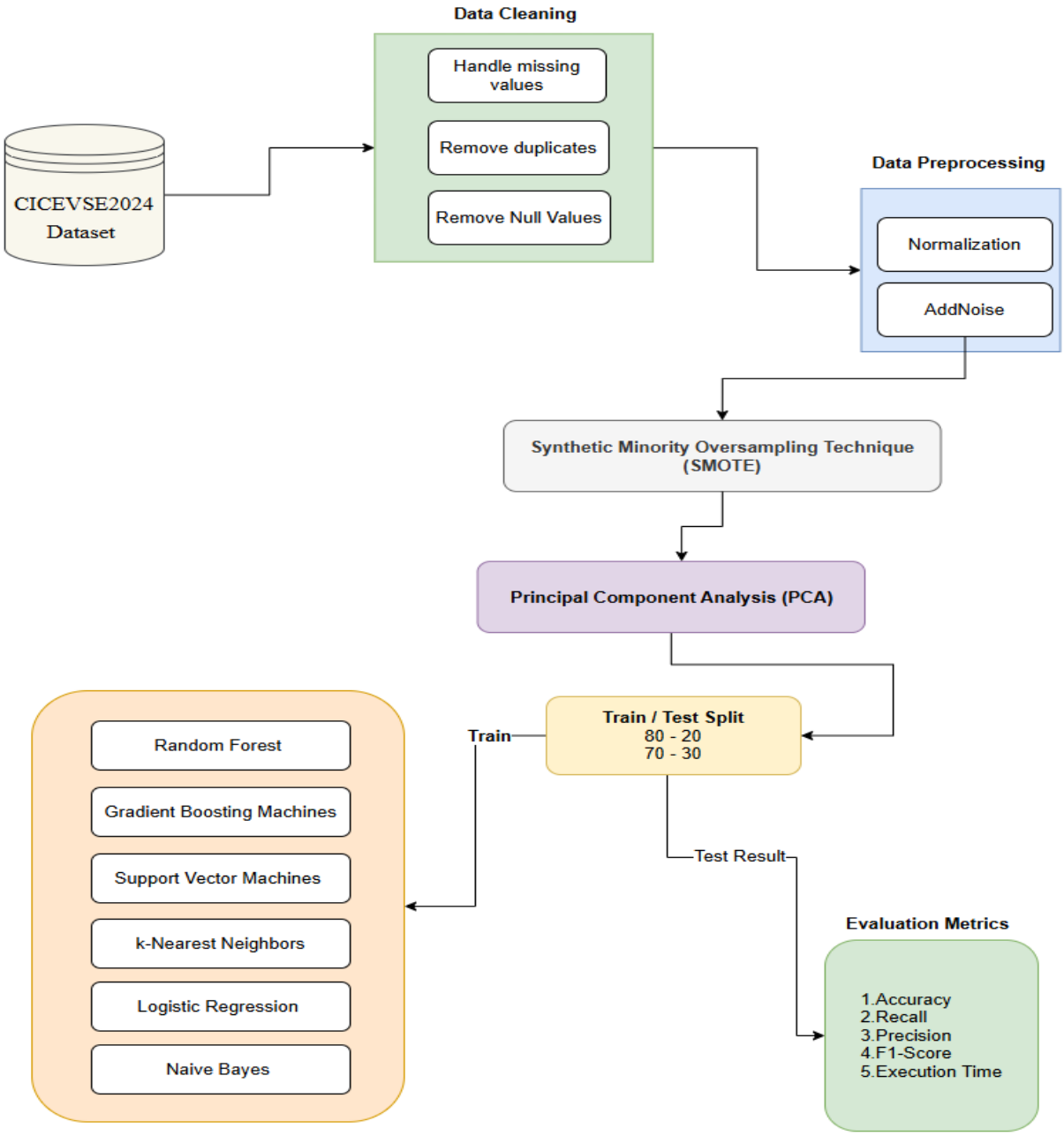


Figure 3.1 System Architecture for EV charging station attack detection.

3.1 CICEVSE2024 Dataset

The dataset used in this study, called CICEVSE2024, was created by researchers at the Canadian Institute for Cybersecurity (CIC) to help researchers improve security for EV charging stations. The data includes information collected from a real EV charging station setup, which included devices like Raspberry Pi and used common communication systems such as OCPP and ISO 15118. The dataset records how the charging station behaves during normal operations and during different types of cyberattacks.

The dataset includes attacks like TCP Port Scan, Service Version Detection, OS Fingerprinting, Aggressive Scan, SYN Stealth Scan, Vulnerability Scan, Slowloris Scan, UDP Flood, ICMP Flood, PSHACK Flood, ICMP Fragmentation, TCP Flood, SYN Flood, SynonymousIP Flood, Cryptojacking, Backdoor. Those scenarios were evaluated on idle and on charge mode of the station.

The data also contains information about the charging station power consumption, internal hardware functioning, and network activity. This is important to use for EV-Based Interactions as they can be used to detect where the attack may happen and what are the interactions that lead to it, providing information that is useful when training ML models to detect attacks [4].

3.1.1 Reconnaissance

Reconnaissance is when attackers collect information about a system and find the gaps. In the CICEVSE2024 dataset, this includes actions like checking open ports, finding running services, and gathering system details. For EV charging stations, these tests were done in both idle and charge mode to see if the system can detect and stop these early attempts [4], [17].

3.1.2 SYN floods

SYN flood is an attack in which someone sends lots of fake requests to connect to a server and doesn’t complete the handshake. This overloads the server and prevents it from functioning correctly. With EVSE, such as this type of attack can hang the system and make it difficult or even impossible for real road users to charge their vehicles [4], [18].

3.1.3 UDP floods

UDP flood is a type of attack where the attacker sends a lot of fake data to a server through UDP, which is a kind of communication method. This overwhelms the server, and it cannot respond to real users. In the case of EVSE, a UDP flood can be aimed at the station's network so that the station cannot communicate correctly, causing unavailability of service for users and interrupting vehicle charging [4], [19].

3.1.4 Backdoor Attack

Backdoor attack deals with an unauthorized user secretly entering a system by exploiting vulnerabilities or using hidden access points. It enables an attacker to gain control of the system in EVSE and perform malicious activities, including downloading harmful files, encrypting data, and altering system settings. This can cause significant problems, such as preventing users from charging their vehicles and disrupting the station's normal operations [4], [20].

3.2 Dataset Cleaning

Data cleaning means finding and fixing problems like missing information, duplicate entries, or data that isn’t useful [21]. During dataset cleaning, an error occurred while loading the dataset into WEKA, as shown in Figure 3.2. The issue happened because the dataset combined data of two different datasets. The top rows had 911 features, whereas middle rows had 915 features. This mismatch, along with empty fields and unexpected string values, caused the error.

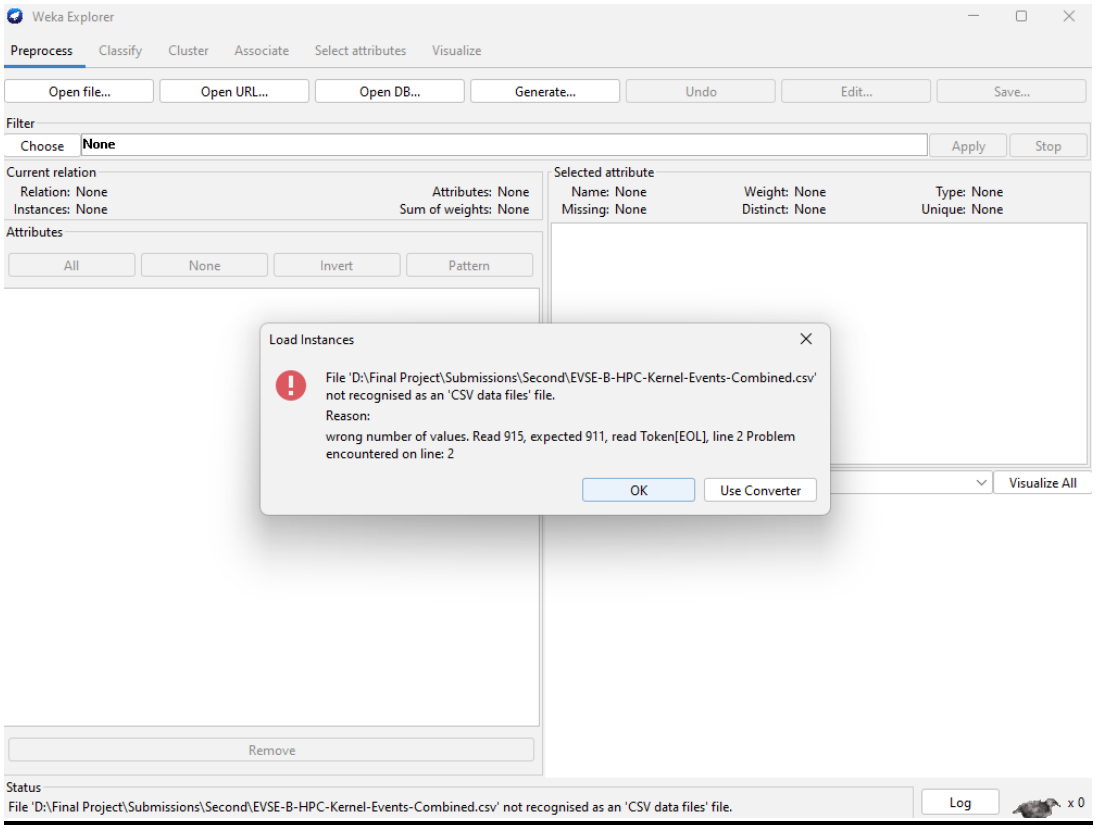


Figure 3.2 CSV File Recognition Error.

To fix this error, the dataset was split into two parts to compare their column names. First column names were extracted from both dataset than carefully verified and picked only matched, as shown in Figure 3.3. 911 columns were found to match between the two datasets out of 915. These matching columns were selected for further use.

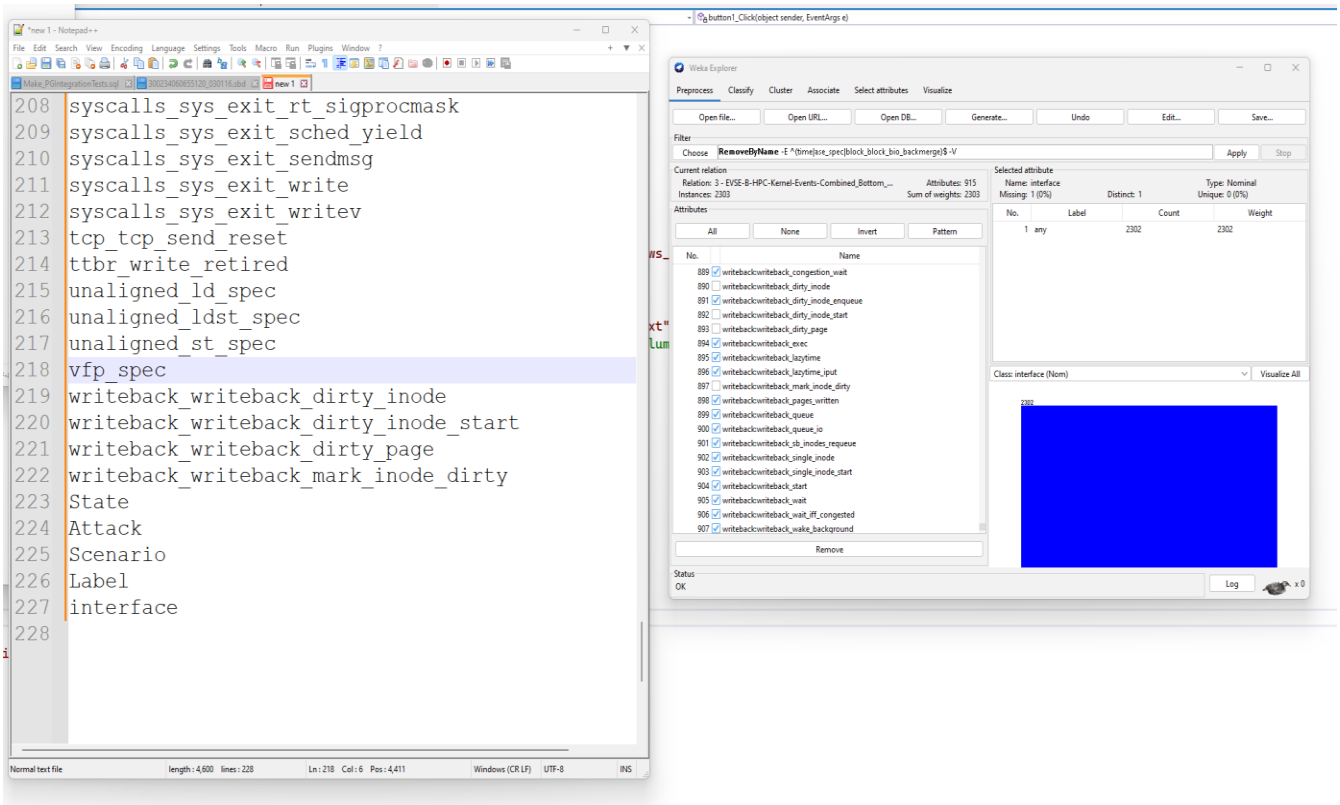
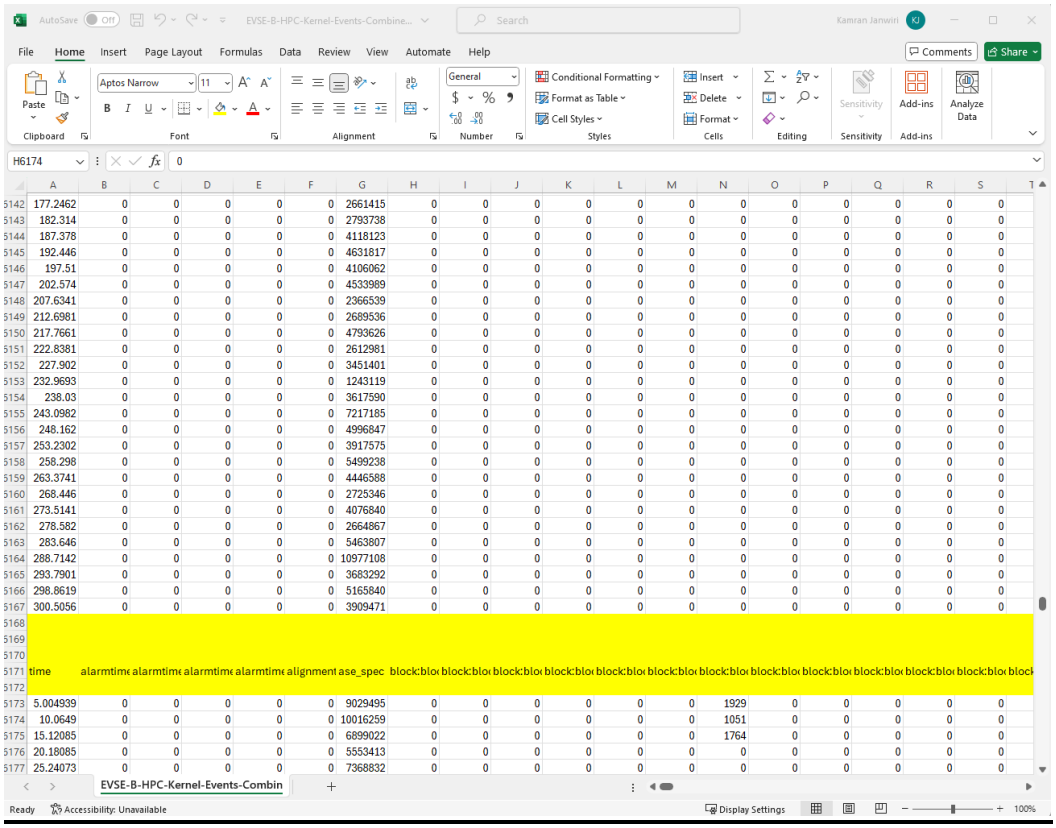


Figure 3.3 Selected common features.

Furthermore, as shown in Figure 3.4, five rows had string values or mostly empty fields. These rows were causing problems as well, so they were removed. These steps were important to clean the dataset and load it properly into WEKA for further analysis.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
5142	177.2462	0	0	0	0	0	2661415	0	0	0	0	0	0	0	0	0	0	0	0
5143	182.314	0	0	0	0	0	2793738	0	0	0	0	0	0	0	0	0	0	0	0
5144	187.378	0	0	0	0	0	4118123	0	0	0	0	0	0	0	0	0	0	0	0
5145	192.446	0	0	0	0	0	4631817	0	0	0	0	0	0	0	0	0	0	0	0
5146	197.51	0	0	0	0	0	4106062	0	0	0	0	0	0	0	0	0	0	0	0
5147	202.574	0	0	0	0	0	4533989	0	0	0	0	0	0	0	0	0	0	0	0
5148	207.6341	0	0	0	0	0	2366539	0	0	0	0	0	0	0	0	0	0	0	0
5149	212.6981	0	0	0	0	0	2689536	0	0	0	0	0	0	0	0	0	0	0	0
5150	217.7661	0	0	0	0	0	4793626	0	0	0	0	0	0	0	0	0	0	0	0
5151	222.8381	0	0	0	0	0	2612981	0	0	0	0	0	0	0	0	0	0	0	0
5152	227.902	0	0	0	0	0	3451401	0	0	0	0	0	0	0	0	0	0	0	0
5153	232.9693	0	0	0	0	0	1243119	0	0	0	0	0	0	0	0	0	0	0	0
5154	238.03	0	0	0	0	0	3617590	0	0	0	0	0	0	0	0	0	0	0	0
5155	243.0982	0	0	0	0	0	7217185	0	0	0	0	0	0	0	0	0	0	0	0
5156	248.162	0	0	0	0	0	4996847	0	0	0	0	0	0	0	0	0	0	0	0
5157	253.2302	0	0	0	0	0	3917575	0	0	0	0	0	0	0	0	0	0	0	0
5158	258.298	0	0	0	0	0	5499238	0	0	0	0	0	0	0	0	0	0	0	0
5159	263.3741	0	0	0	0	0	4446588	0	0	0	0	0	0	0	0	0	0	0	0
5160	268.446	0	0	0	0	0	2725346	0	0	0	0	0	0	0	0	0	0	0	0
5161	273.5141	0	0	0	0	0	4076840	0	0	0	0	0	0	0	0	0	0	0	0
5162	278.582	0	0	0	0	0	2664867	0	0	0	0	0	0	0	0	0	0	0	0
5163	283.646	0	0	0	0	0	5463807	0	0	0	0	0	0	0	0	0	0	0	0
5164	288.7142	0	0	0	0	0	10977108	0	0	0	0	0	0	0	0	0	0	0	0
5165	293.7901	0	0	0	0	0	3683292	0	0	0	0	0	0	0	0	0	0	0	0
5166	298.8619	0	0	0	0	0	5165840	0	0	0	0	0	0	0	0	0	0	0	0
5167	300.5056	0	0	0	0	0	3909471	0	0	0	0	0	0	0	0	0	0	0	0
5168																			
5169																			
5170																			
5171	time	alarmtime	alarmtime	alarmtime	alarmtime	alignment	ase_spec	block:blo	block:blo	block:blo	block:blo	block:blo	block:blo	block:blo	block:blo	block:blo	block:blo	block:blo	block:blo	block:blo	block:blo
5172																			
5173	5.004939	0	0	0	0	0	9029495	0	0	0	0	0	0	1929	0	0	0	0	0
5174	10.0649	0	0	0	0	0	10016259	0	0	0	0	0	0	1051	0	0	0	0	0
5175	15.12085	0	0	0	0	0	6899022	0	0	0	0	0	0	1764	0	0	0	0	0
5176	20.18085	0	0	0	0	0	5553413	0	0	0	0	0	0	0	0	0	0	0	0
5177	25.24073	0	0	0	0	0	7368832	0	0	0	0	0	0	0	0	0	0	0	0

Figure 3.4 The Problematic Rows in the dataset.

3.3 Dataset Preprocessing

Dataset preprocessing is important for making classification tasks work well. In this project, WEKA was used to apply several preprocessing steps to make the dataset clean, consistent, and ready for reliable model performance [21].

The following WEKA filters were used in this project:

1. **Replace Missing Values**

   ReplaceMissingValues filter was used to fix missing data in a dataset. This filter replaces missing category values (nominal) with the most common value in that column. For missing numbers, it uses the average value of the column [21].

2. **Remove Duplicates Values**

   To remove the duplicate values WEKA does provide filter called “RemoveDuplicate” it handles duplicate data in the dataset, this filter ensured that all duplicate rows were removed [21].

3. **Remove Irrelevant Values**

   The "RemoveUseless" filter was applied to eliminate unnecessary or irrelevant attributes in the dataset. As it was unclear which attributes might not contribute to the analysis, the filter removed those with little or no variation. A total of 685 attributes were eliminated after applying this filter [21], [22].

4. **Normalization**

   This filter was applied to change the numbers in a dataset so they all fit within the same range, usually between 0 and 1. In a dataset, some numbers can be very small while others are very big. This can cause problems for ML models. Normalizing makes sure that all numbers are scaled to the same range, which helps the model work better.

$$Xn = \frac{\mathrm{X} - \mathrm{Xmin}}{Xmax - Xmin}$$

- Xn : The new number.
- X: The original number.
- $Xmin$: The smallest number in the dataset.
- $Xmax$: The largest number in the dataset.

This formula makes the smallest number in the dataset become 0, the largest number become 1, and all other numbers fall between 0 and 1. For example, if the numbers in a dataset are between 10 and 50, the formula will change 10 to 0, 50 to 1, and a number like 30 will become 0.5. Normalization helps ML models work better by putting all the numbers on the same scale [21], [23].

3.4 Synthetic Minority Oversampling Technique (SMOTE)

SMOTE is a method used to fix problems when one class in a dataset has much more data than the other. Instead of copying the smaller group of data, it creates new, realistic data by mixing similar points [5], [6].

For example, Figure 3.5 shows that the dataset originally had 8,464 entries: 6,166 were attack, and only 2,302 were benign. This imbalance could make the model focus more on the attack class and ignore the benign class. After using SMOTE, the dataset was balanced to 6,166 attack and 4,604 benign entries. This balance helps the model learn about both classes equally, making it more accurate and reliable.

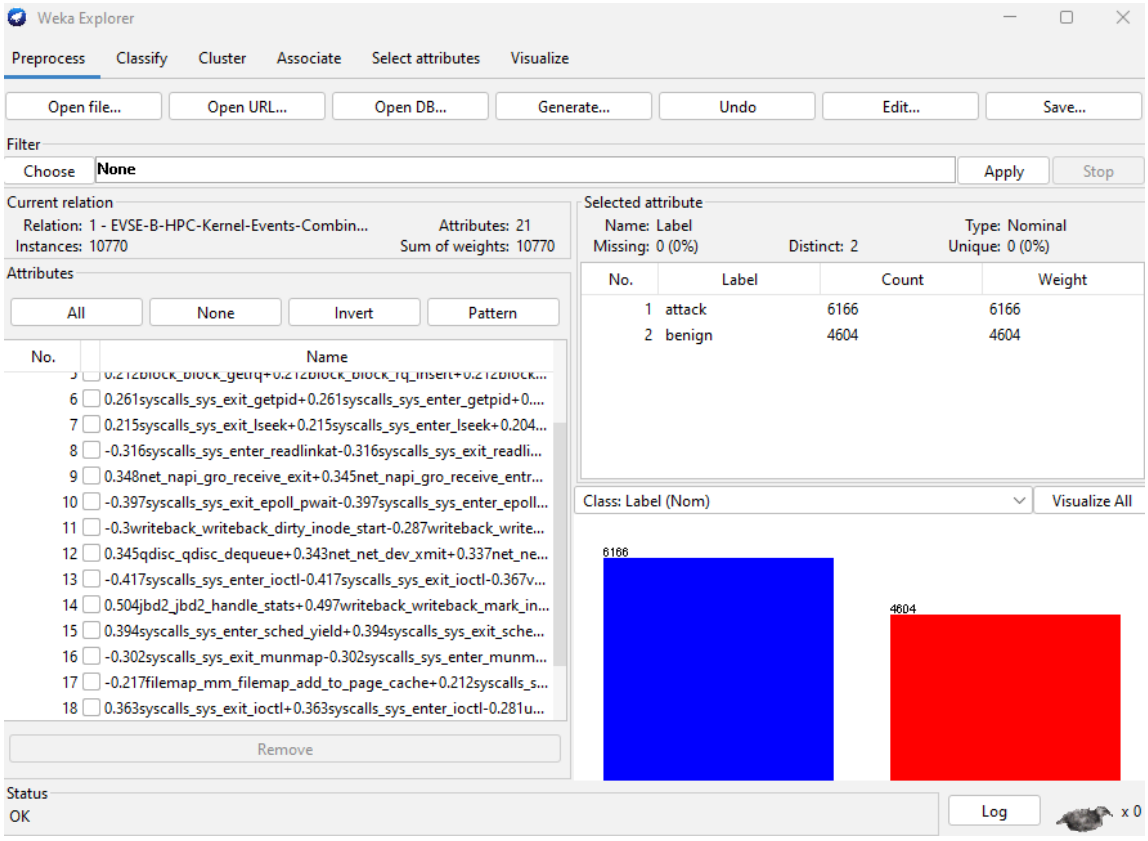


Figure 3.5 After applying SMOTE.

Figure 3.6 shows the WEKA Package Manager. Since SMOTE is not built into WEKA, so it needs to be installed it through Tools > Package Manager.

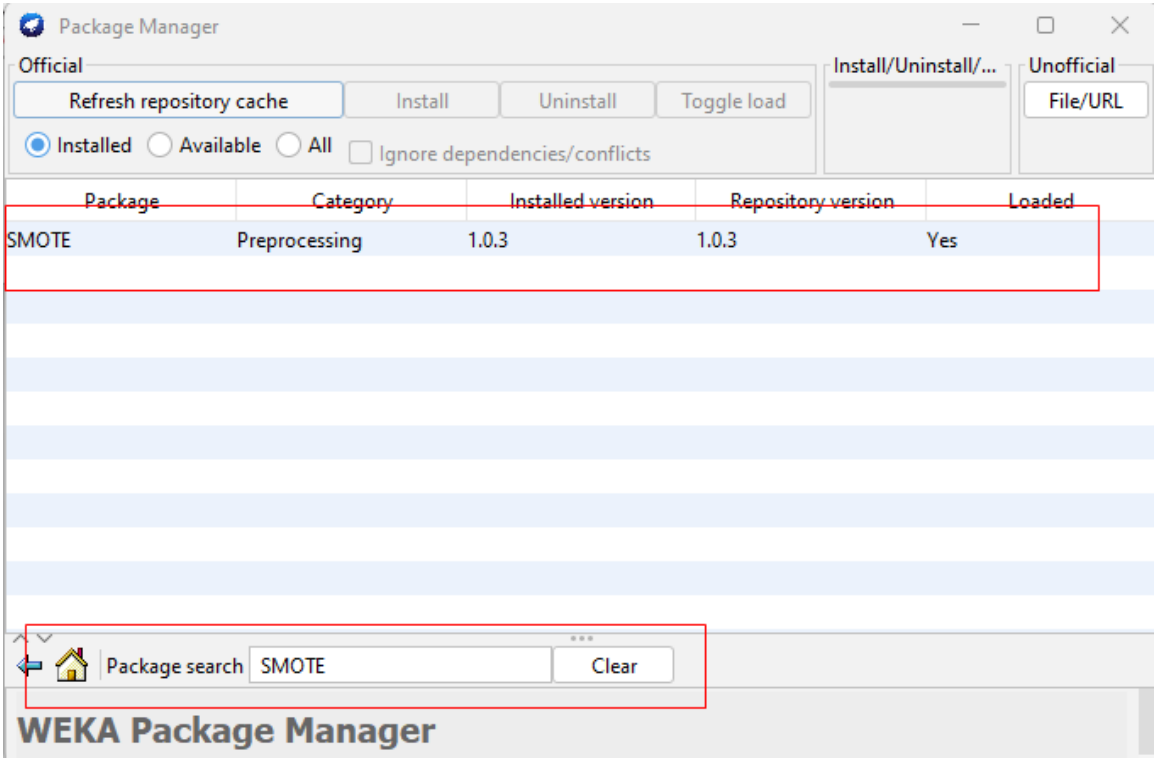


Figure 3.6 WEKA Package Manager.

3.5 Principal Component Analysis (PCA)

PCA is used to simplify the dataset by reducing unnecessary features. The best part of PCA it keeps only the most relevant information while removing redundant data [7], [8]. Before applying PCA, the dataset had 221 features, but after PCA, it was reduced to 21 key features as shown in Figure 3.6. This allows the model to focus on the most useful data, making it faster, more efficient, and less likely to overfit.

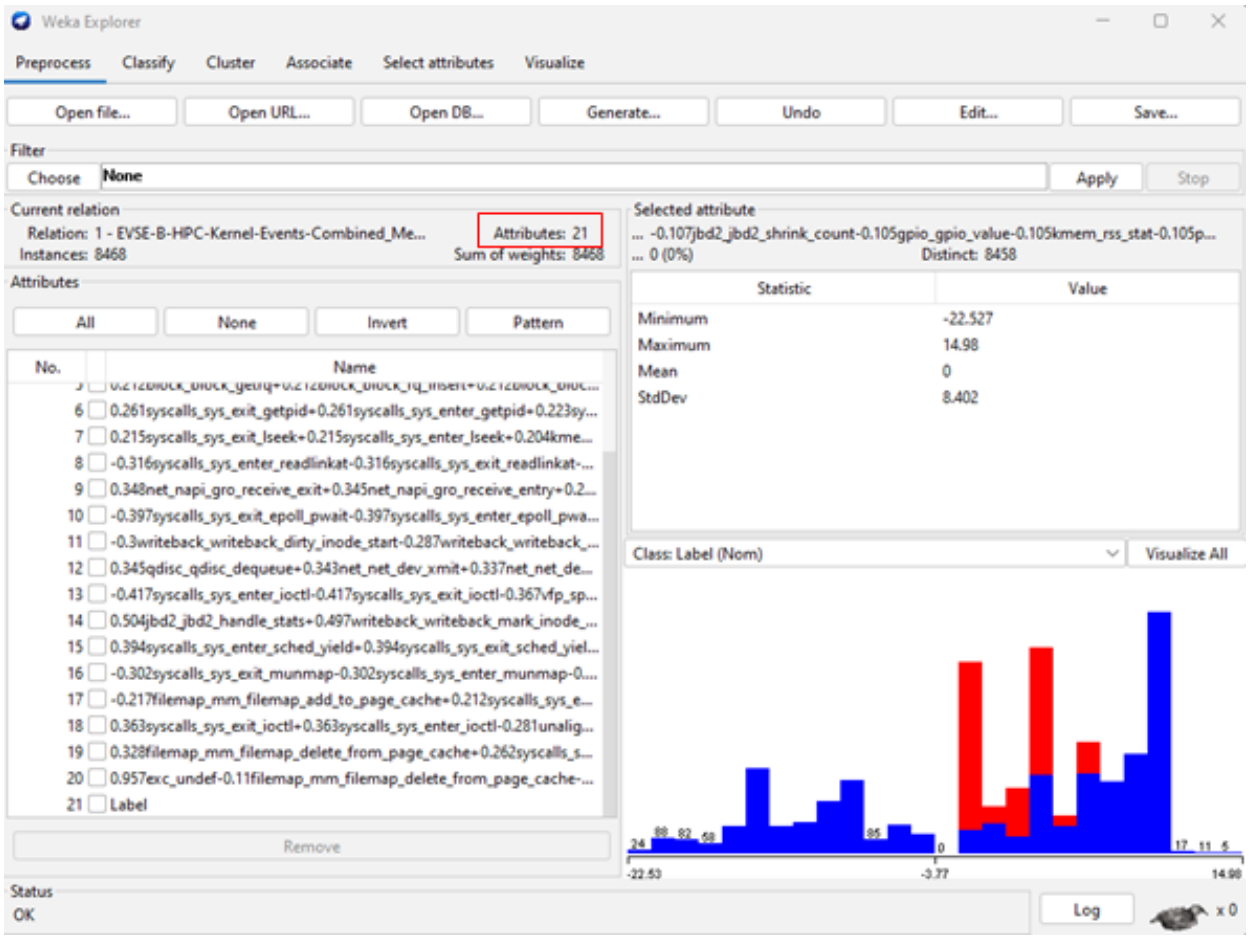


Figure 3.6 After applying PCA.

3.6 ML Algorithms

3.6.1 Random Forest (RF)

RF is a ML method that uses many decision trees to make predictions. Each decision tree is trained on different parts of the data and makes its own prediction based on certain conditions. For example, one tree might predict attack, while another predicts benign. The final result comes from a majority vote of all the trees. This means the prediction chosen by most trees is the one used as the final answer. Using multiple trees helps make RF more accurate and less likely to make mistakes [24].

3.6.2 Gradient Boosting Machines (GBM)

GBM is a ML method that combines several decision trees to improve predictions. It works step by step, where each new tree focuses on fixing the mistakes made by the previous ones. This process makes the model better and more accurate over time. GBM is good for tasks like classification and regression. It is also effective at handling unbalanced data, where one category might have much more data than another [25].

3.6.3 Support Vector Machines (SVM)

SVM is a ML method that helps to sort data into different groups. It tries to find the best line or boundary that separates the groups, making sure there is as much space as possible between them. The closest points to the line are called support vectors, and they help create the boundary. If the data is too complicated to be separated with a straight line, SVM can use a special method called the kernel trick to make it easier to separate the groups [26].

3.6.4 K-Nearest Neighbors (KNN)

KNN is a ML method that used to classify data into groups. It works by looking at the "K" closest data points to a new data point and then deciding which group the new point belongs to based on the most common group among those K neighbors. For example, if 3 out of 5 neighbors belong to Group A, the new point will also be assigned to Group A. K-NN doesn’t need to learn from the data in advance; it simply looks at the data points around it to make decisions. The value of "K" is chosen before, and it controls how many neighbors to consider when making the decision [27].

3.6.5 Logistic Regression (LR)

LR is a ML method used to classify data into two groups, like yes or no. It works by finding a relationship between the input data and the outcome, it creates a line or curve to separate the two groups. Instead of predicting exact values, it predicts the probability of a data point belonging to a certain group. If the probability is higher than a certain threshold, the point is classified into one group, and if it is lower, it goes into the other group [28].

3.6.6 Naive Bayes (NB)

NB is a ML method that performs ML based on Bayes Theorem. While this model assumes independence between features, an assumption often referred to as the "naive" assumption of the model, in real data this is not often the case. The model is, however, very easy and quick to utilize. NB calculates a probability for each class given the input using the different features provided. It then predicts which one has the highest probability. It works well on both binary and multi-class classification tasks [29].

Chapter 4: Performance Evaluation of ML Models for EV Charging Station Attack Detection

In this chapter, the performance of six supervised ML algorithms, namely RF, GBM, SVM, KNN, LR, and NB are evaluated. The results were obtained using a personal laptop with specifications detail listed in Table 4.1.

Table 4.1 System Specification.

Component	**Details**
Brand	Lenovo
Model Number	20LAS0A000
OS Version	Windows 11 Home 64 bit
Processor	Intel Core i5-8350U @ 1.70GHz
Memory Capacity	16 GB DDR4
CPU Speed	1.70 GHz (Base), up to 3.60 GHz
Core Count	4
Weka Version	3.8.6

4.1 Evaluation Metrics

Precision tells us how many of the predicted attacks are actually correct. It shows how accurate the model is when it predicts something as an attack. To calculate precision, we divide the number of correctly predicted attacks (True Positives) by the total number of attack predictions, including both correct (True Positives) and wrong ones (False Positives) [30], [31].

$$Precision = \frac{True\ Positive\ (tp)}{True\ Positive\ (tp) + False\ Positive\ (fp)}$$

Recall tells us how many of the real attacks the model successfully finds. It measures the model's ability to catch all actual attacks. To calculate recall, we divide the total number of attacks correctly identified attacks (True Positives) by the total number of actual attacks, which includes both detected (True Positives) and missed ones (False Negatives) [30], [31].

$$Recall = \frac{True\ Positive\ (tp)}{True\ Positive\ (tp) + False\ Negative\ (fn)}$$

Accuracy tells us how often the model makes correct predictions. It considers both correctly predicted attacks (True Positives) and correctly predicted benign cases (True Negatives) and divides this by the total number of all instances, including both correct and incorrect predictions [30], [31].

$$Accuracy = \frac{True\ Positive\ (tp) + True\ Negative\ (tn)}{True\ Positive\ (tp) + False\ Positive\ (fp) + True\ Negative\ (tn) + False\ Nagtive\ (fn)}$$

F1-Score combines precision and recall into one value, balancing both. It's especially helpful for imbalanced datasets and gives equal importance to accuracy in predictions and finding actual positives [30], [31].

$$F1 - Score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

Execution Time refers to how long it takes to train and test the model.

To Evaluate the model, we mainly will focus on Accuracy and Execution time.

4.2 Models Performance Using 70/30% Split with 21 Features

Table 4.2 shows the models performance with 21 features, and Figure 4.1 shows accuracy in a graph. RF had the highest accuracy at 96%. GBM followed with 92% accuracy, while SVM and KNN achieved 94.3% and 94.1%, respectively. LR reached 94.9% accuracy. NB was the fastest model but had the lowest accuracy at 87.2%. Overall, RF performed the best, and KNN was the quickest.

Table 4.2 Models results with 70/30% Split (21 Features).

Model	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)	Execution Time (s)
Random Forest	96	95.5	96	96	8.8
Gradient Boosting Machines	92	92	91.6	92	0.8
Support Vector Machines	94.3	94.4	94.3	94.3	1.22
K-Nearest Neighbors	94.1	94.1	94.1	94.1	<0.01
Logistic Regression	94.9	94.9	94.9	94.9	0.44
Naive Bayes	87.2	87.4	87.2	87.3	0.04

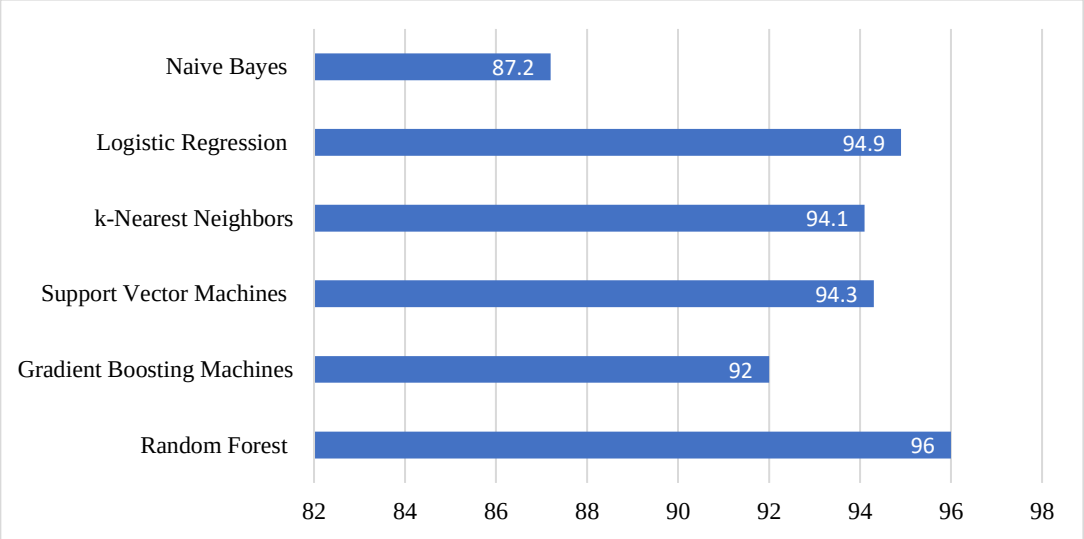


Figure 4.1 Models Accuracy with 70/30% Split (21 Features)

4.3 Models Performance Using 70/30% Split with 15 Features

Table 4.3 shows the models performance with 15 features, and Figure 4.2 shows accuracy in a graph. RF had the highest accuracy at 95.9% but took the longest time (7.25 seconds). GBM followed with 93.1% accuracy, while LR and KNN achieved 94.5% and 94%, respectively. NB was the fastest model at 0.04 seconds, with an accuracy of 93.4%. Overall, RF performed the best, and NB was the quickest.

Table 4.3 Models results with 70/30% Split (15 Features).

Model	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)	Execution Time (s)
Random Forest	95.9	95.9	95.9	95.9	7.25
Gradient Boosting Machines	93.1	93.2	93.1	93.1	0.61
Support Vector Machines	92.9	92.9	92.9	92.9	0.57
K-Nearest Neighbors	94	94	94	94	1.16
Logistic Regression	94.5	94.5	94.5	94.5	0.21
Naive Bayes	93.4	93.6	93.4	93.5	0.04

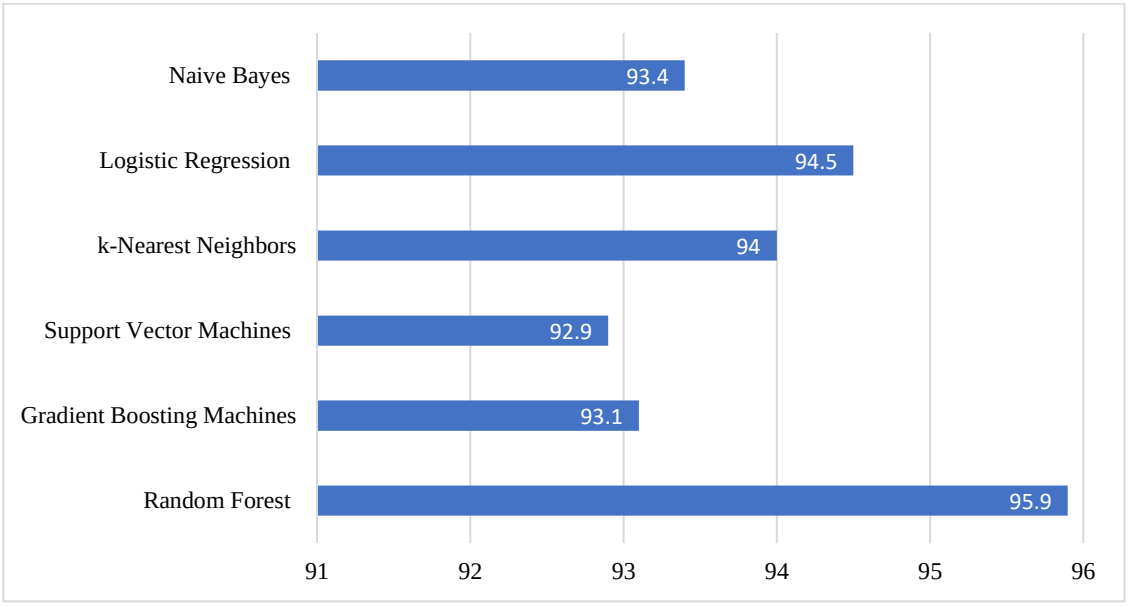


Figure 4.2 Models Accuracy with 70/30% Split (15 Features)

4.4 Models Performance Using 70/30% Split with 10 Features

Table 4.4 shows the models performance with 10 features, and Figure 4.3 shows accuracy in a graph. RF had the highest accuracy at 95.9%. GBM followed with 93.5% accuracy, while SVM achieved 93.1%. KNN and LR both had 94% accuracy. NB was best at execution time, taking only 0.02 seconds, with an accuracy of 93.4%. Overall, RF performed the best, and KNN was the quickest.

Table 4.4 Models results with 70/30% Split (10 Features).

Model	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)	Execution Time (s)
Random Forest	95.9	95.9	95.9	95.9	0.21
Gradient Boosting Machines	93.5	93.6	93.5	93.5	0.41
Support Vector Machines	93.1	93.3	93.1	93.1	1.1
K-Nearest Neighbors	94	94	94	94	<0.01
Logistic Regression	94	94.1	94	94	0.15
Naive Bayes	93.4	93.5	93.4	93.4	0.02

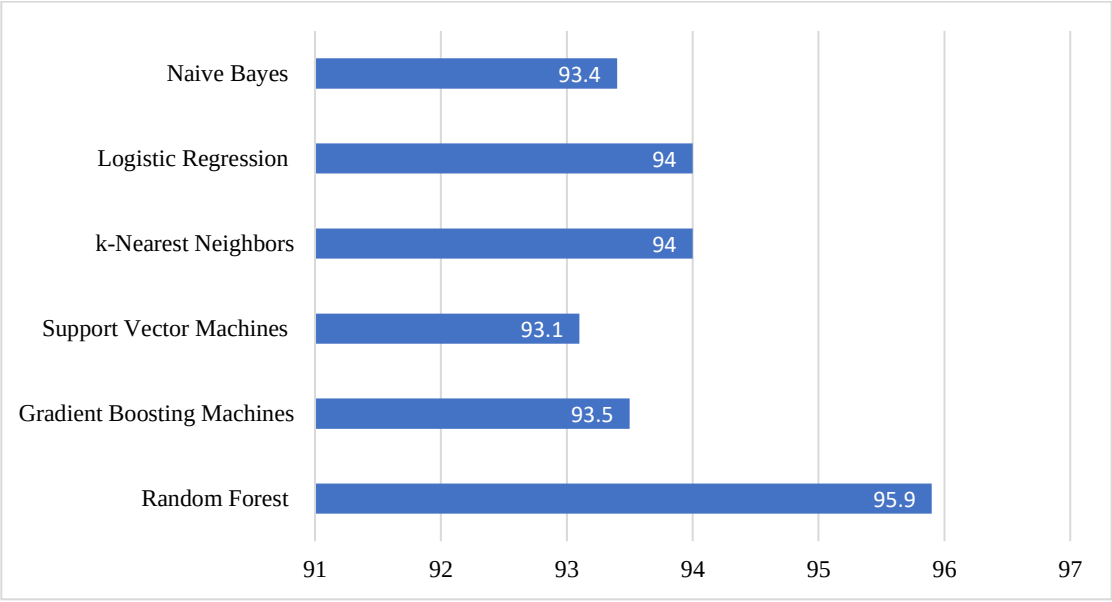


Figure 4.3 Models Accuracy with 70/30% Split (10 Features)

4.5 Models Performance Using 70/30% Split with 5 Features

Table 4.5 shows the models performance with 5 features, and Figure 4.4 shows accuracy in a graph. RF had the highest accuracy at 95.8%. KNN and LR followed with 93.4% and 93.1% accuracy, respectively. SVM achieved 92.2%, while GBM had 89.2% accuracy. NB was the fastest model, taking only 0.01 seconds, but had the lowest accuracy at 90.34%. Overall, RF performed the best, and KNN was the quickest.

Table 4.5 Models results with 70/30% Split (5 Features).

Model	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)	Execution Time (s)
Random Forest	95.8	95.2	95.1	95.2	0.21
Gradient Boosting Machines	89.2	89.4	89.2	89.2	0.2
Support Vector Machines	92.2	92.6	92.2	92.3	0.39
K-Nearest Neighbors	93.4	93.5	93.4	93.4	<0.01
Logistic Regression	93.1	93.3	93.1	93.1	0.11
Naive Bayes	90.34	91	90.3	90.4	0.01

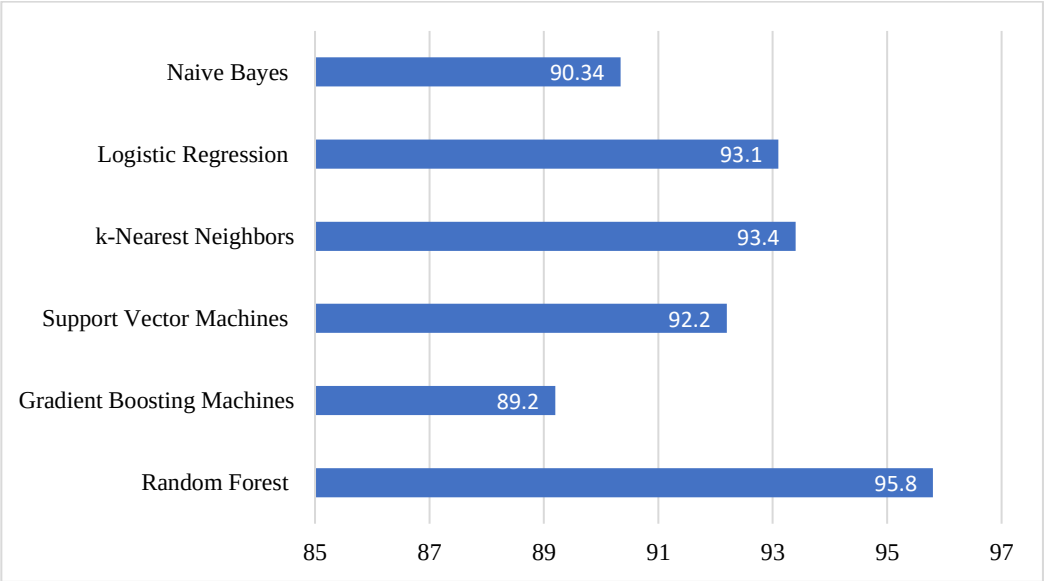


Figure 4.4 Models Accuracy with 70/30% Split (5 Features)

4.6 Models Performance Using 10-Fold Cross-Validation with 21 Features

Table 4.6 shows the performance of models using 10-fold cross-validation with 21 features, and Figure 4.5 shows their accuracy in a graph. RF had the best accuracy at 96.7% but took the longest time to run 9.8 seconds. SVM followed with 94.8% accuracy, while LR achieved 95.4% accuracy and had a short execution time of 0.6 seconds. KNN also performed well, with 94.3% accuracy and almost no execution time. GBM scored 92.4% accuracy, and NB had the lowest accuracy at 84.7%, but it was very good at execution time, taking only 0.1 seconds. Overall, RF was the most accurate, while KNN was the quickest.

Table 4.6 Models results with 10-Fold Cross-Validation (21 Features).

Model	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)	Execution Time (s)
Random Forest	96.7	96.7	96.7	96.7	9.8
Gradient Boosting Machines	92.4	92.5	92.4	92.5	0.8
Support Vector Machines	94.8	94.9	94.8	94.8	1.8
K-Nearest Neighbors	94.3	94.3	94.3	94.3	<0.01
Logistic Regression	95.4	95.5	95.4	95.4	0.6
Naive Bayes	84.7	84.9	84.7	84.7	0.1

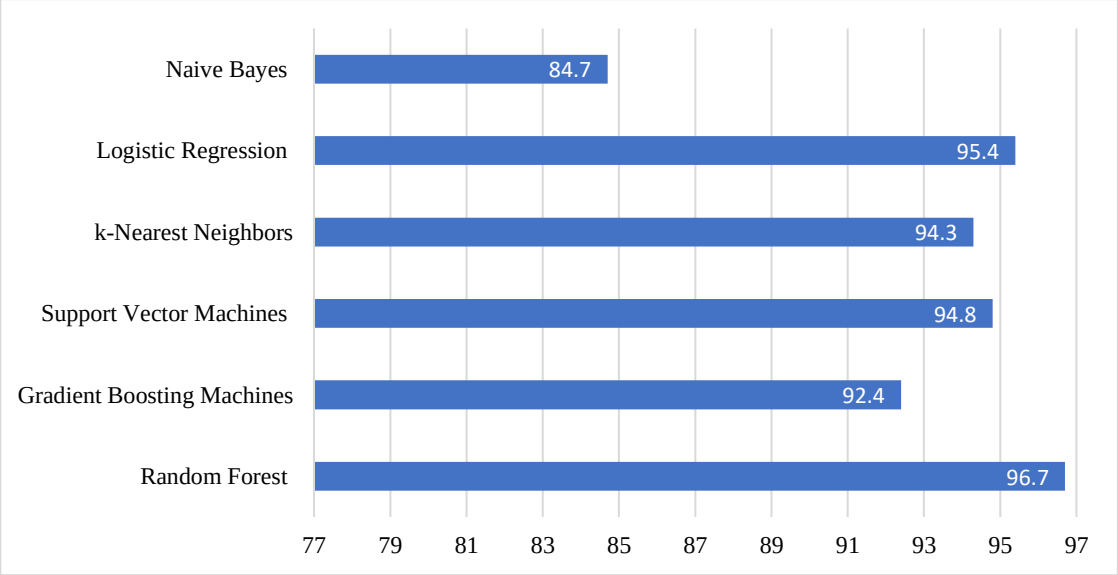


Figure 4.5 Models Accuracy with 10-Fold Cross-Validation (21 Features).

4.7 Models Performance Using 10-Fold Cross-Validation with 15 Features

Table 4.7 shows the performance of models using 10-fold cross-validation with 15 features, and Figure 4.6 shows their accuracy in a graph. RF had the best accuracy at 96.7% but took the longest time to run 6.8 seconds. LR followed with 94.7% accuracy and a short execution time of 0.2 seconds. KNN performed well with 94.2% accuracy and almost no execution time. SVM and GBM both achieved 93.8% accuracy. NB had the lowest accuracy at 93.5% but was the fast model, taking only 0.1 seconds. Overall, RF performed the best, and KNN was the quickest.

Table 4.7 Models results with 10-Fold Cross-Validation (15 Features).

Model	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)	Execution Time (s)
Random Forest	96.7	96.7	96.7	96.7	6.8
Gradient Boosting Machines	93.8	93.8	93.8	93.8	0.5
Support Vector Machines	93.8	94	93.8	93.8	0.6
K-Nearest Neighbors	94.2	94.2	94.2	94.2	<0.01
Logistic Regression	94.7	94.8	94.7	94.7	0.2
Naive Bayes	93.5	93.6	93.5	93.5	0.1

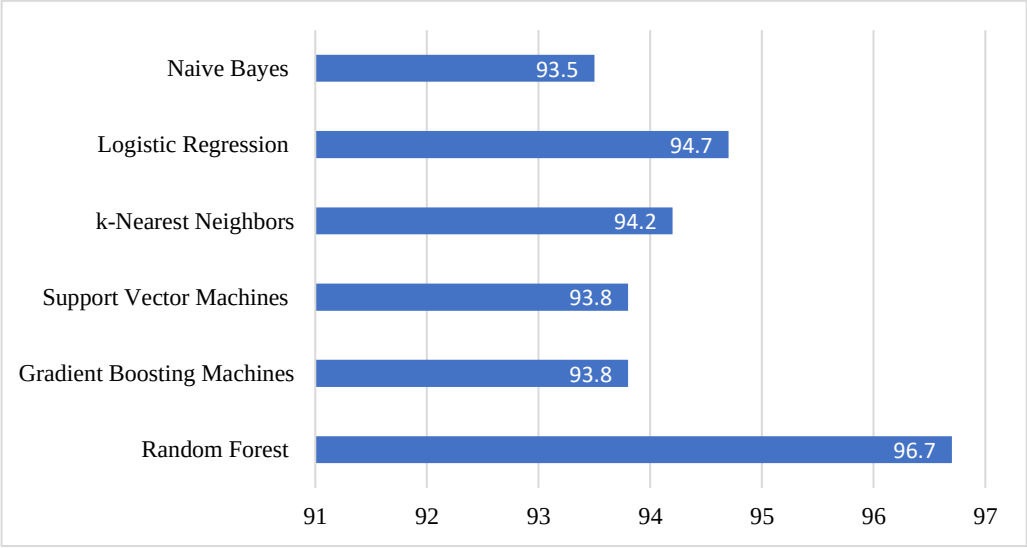


Figure 4.6 Models Accuracy with 10-Fold Cross-Validation (15 Features).

4.8 Models Performance Using 10-Fold Cross-Validation with 10 Features

Table 4.8 shows the performance of models using 10-fold cross-validation with 10 features, and Figure 4.7 displays their accuracy in a graph. RF had the best accuracy at 96.7% but took the longest time to run 10.5 seconds. LR and KNN performed well with accuracies of 94.3% and 94.2%, respectively, with very short execution times of 0.2 seconds and 0.01 seconds. GBM and SVM achieved accuracies of 93.7% and 93.4%, respectively. NB had the lowest accuracy at 93.3% but was fast model, taking only 0.03 seconds. Overall, RF performed the best, and KNN was the quickest.

Table 4.8 Models results with 10-Fold Cross-Validation (10 Features).

Model	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)	Execution Time (s)
Random Forest	96.7	96.7	96.7	96.7	10.5
Gradient Boosting Machines	93.7	93.7	93.7	93.7	0.6
Support Vector Machines	93.4	93.7	93.4	93.5	0.7
K-Nearest Neighbors	94.2	94.2	94.2	94.2	0.01
Logistic Regression	94.3	94.4	94.3	94.3	0.2
Naive Bayes	93.3	93.4	93.3	93.3	0.03

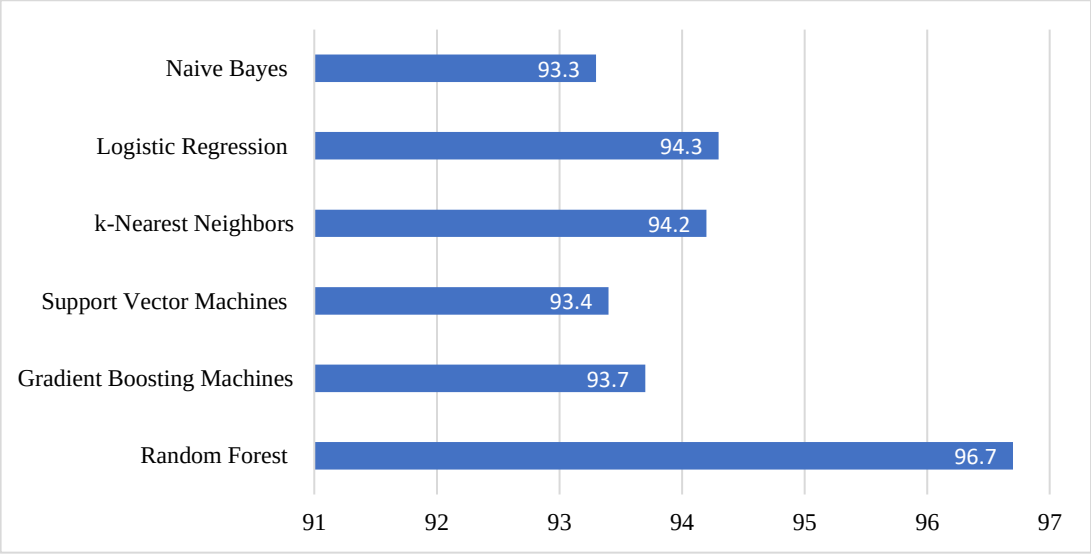


Figure 4.7 Models Accuracy with 10-Fold Cross-Validation (10 Features).

4.9 Models Performance Using 10-Fold Cross-Validation with 5 Features

Table 4.9 shows the performance of models using 10-fold cross-validation with 5 features, and Figure 4.8 displays their accuracy in a graph. RF achieved the highest accuracy at 96.5%, followed by KNN with 93.9% accuracy. LR also performed well with 93.3% accuracy, while SVM scored 92.5%, and GBM had 89.2% accuracy. NB had the lowest accuracy at 90.1% but was the fastest model, taking only 0.01 seconds. Overall, RF performed the best, and KNN was the quickest.

Table 4.9 Models results with 10-Fold Cross-Validation (5 Features).

Model	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)	Execution Time (s)
Random Forest	96.5	96.5	96.5	96.5	5.6
Gradient Boosting Machines	89.2	89.4	89.2	89.2	0.2
Support Vector Machines	92.5	92.9	92.5	92.5	0.4
K-Nearest Neighbors	93.9	93.9	93.9	93.9	<0.01
Logistic Regression	93.3	93.6	93.3	93.4	0.1
Naive Bayes	90.1	90.9	90.1	90.2	0.01

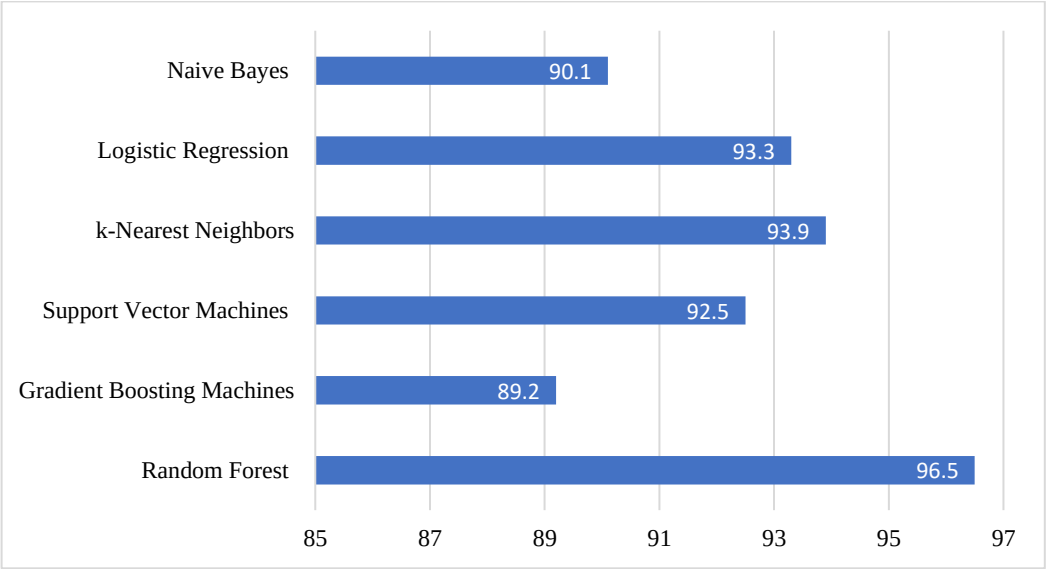


Figure 4.8 Models Accuracy with 10-Fold Cross-Validation (5 Features).

4.8 Models Performance Using 2-Fold Cross-Validation with 10 Features

Table 4.10 shows the performance of models using 2-fold cross-validation with 10 features, and Figure 4.8 displays their accuracy in a graph. RF had the best accuracy at 96.6% but took the longest time to run 6.2 seconds. LR and KNN performed well with accuracies of 94.3% and 94.2%, respectively, with very short execution times of 0.3 seconds and 0.01 milliseconds. GBM and SVM achieved accuracies of 94 % and 93.1%, respectively. NB had the lowest accuracy at 93.5% but was fast model, taking only 0.1 seconds. Overall, RF performed the best, and KNN was the quickest.

Table 4.10 Models results with 2-Fold Cross-Validation (10 Features).

Model	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)	Execution Time (s)
Random Forest	96.6	96.6	96.6	96.6	6.2
Gradient Boosting Machines	94	94.1	94	94	0.4
Support Vector Machines	93.1	93.4	93.1	93.2	0.7
K-Nearest Neighbors	94.2	94.2	94.2	94.2	0.01
Logistic Regression	94.3	94.3	94.3	94.3	0.3
Naive Bayes	93.5	93.6	93.5	93.5	0.1

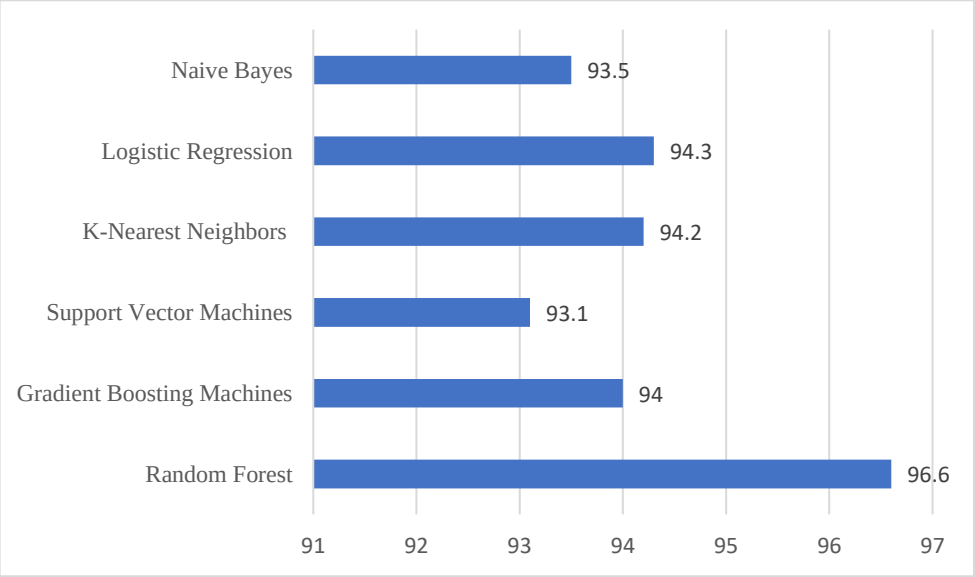


Figure 4.8 Models Accuracy with 2-Fold Cross-Validation (10 Features).

4.10 Top 10 Features

Based on the previous analysis, we used PCA to test different sets of features: 21, 15, 10, and 5. We observed that the accuracy and execution time improved consistently with certain features. Among these, the top 10 features were identified as the most important for model performance. These features are listed in Table 4.11.

Table 4.11 Top 10 Features.

No.	Name
1	0.4640.212block_block_getrq+0.212block_block_rq_insert+0.212block_block_rq_issue+0.212block_block_rq_complete+0.211block_block_...
2	-0.4530.261syscalls_sys_exit_getpid+0.261syscalls_sys_enter_getpid+0.223syscalls_sys_enter_write+0.223syscalls_sys_exit_write+0.215sysca...
3	0.573-0.217filemap_mm_filemap_add_to_page_cache+0.212syscalls_sys_exit_getrandom+0.212syscalls_sys_enter_getrandom-0.204syscall...
4	-0.3930.136cache-references+0.127branch-load-misses+0.126branch-misses+0.126br_mis_pred+0.122raw_syscalls_sys_enter...+0.3880.50...
5	0.8480.957exc_undef-0.11filemap_mm_filemap_delete_from_page_cache-0.102syscalls_sys_enter_faccessat-0.102syscalls_sys_exit_faccess...
6	-0.8670.345qdisc_qdisc_dequeue+0.343net_net_dev_xmit+0.337net_net_dev_start_xmit+0.335net_net_dev_queue-0.287net_napi_gro_recei...
7	-0.8660.149irq_irq_handler_entry+0.149bus_access_periph+0.148irq_softirq_exit+0.148irq_softirq_entry+0.147rpm_rpm_resume...-0.3660.3...
8	0.703-0.397syscalls_sys_exit_epoll_pwait-0.397syscalls_sys_enter_epoll_pwait-0.343syscalls_sys_exit_recvmsg-0.343syscalls_sys_enter_recv...
9	0.4490.363syscalls_sys_exit_ioctl+0.363syscalls_sys_enter_ioctl-0.281unaligned_st_spec+0.242syscalls_sys_enter_rt_sigprocmask+0.242sysc...
10	0.515-0.107jbd2_jbd2_shrink_count-0.105gpio_gpio_value-0.105kmem_rss_stat-0.105pagemap_mm_lru_insertion-0.105skb_skb_copy_data...

4.11 Discussion

The results show that RF had the best accuracy of 96% when using all 21 attributes, but it took the longest time to execute, 8.8 seconds. In comparison, KNN was the fastest model with accuracy of 94.1%, completing in 0 seconds, while NB had the lowest accuracy of 87.2% but was still very quick, taking only 0.04 seconds.

When the attributes were reduced to 15, RF accuracy stayed nearly the same at 95.9%, and its execution time improved to 7.25 seconds. KNN also performed well, maintaining 94% accuracy with a slightly slower time of 1.16 seconds. Interestingly, NB improved its accuracy to 93.4% while keeping the same execution time of 0.04 seconds.

With 10 attributes, RF again maintained high accuracy at 95.9% and showed a big improvement in execution time, dropping to just 0.21 seconds. KNN also have good accuracy at 94% and was extremely fast. NB accuracy stayed steady at 93.4%, but its execution time improved slightly, taking only 0.02 seconds, please note that KNN results are taken with K=1 whereas we have tried in this study K=2 as well but didn’t notice big difference only approximately 0.17%.

Compared to the earlier work using the CICEVSE2024 dataset, this study shows clear improvements. Before, RF had 91.88% accuracy, and KNN had 91.58%, with no details about how long they took to run. In this study, the data was split 70/30 for training and testing, and 10-fold cross-validation was used. The dataset was cleaned, normalized, and balanced using SMOTE, which helped improve accuracy. Execution times were also tracked to better understand model performance. RF's accuracy went up by 5.25%, and KNN's by 2.86%, showing significant progress.

In summary, RF consistently gave the best accuracy, proving to be the most reliable model overall. KNN was the fastest model in most of cases. Reducing the number of attributes to 10 was a good step, as it significantly improved execution time and accuracy, showing the importance of simplifying the dataset.

Chapter 5: Conclusion and Future Work

5.1 Conclusion

In summary, research suggests that ML techniques would be highly effective for identifying and classifying cyber attacks on EVSE. Data cleaning, feature selection, and ML models testing was done using the CICEVSE2024 dataset. KNN reached the highest speed and made it suitable for real-world applications, while high accuracy was reached by RF. SMOTE helped balance the dataset, and PCA reduced the number of features, leading to an improvement in the speed and accuracy of the models. This work demonstrates how ML can strengthen the security of EVSEs. And finally, that we need the right features and a trade-off between accuracy and speed in order for these applications to be useful, as well.

5.2 Future Work

While this work has demonstrated that ML models perform well in terms of detecting and classifying cyber-attacks on EVSE, future research could be pursued to further enhance the security and reliability of EV charging systems.

Real-Time Application Scenarios

This work should be extended further toward real-world application in actual EV charging stations. For example, the Random Forest based intrusion detection system can be used for monitoring and real-time threat detection. However, deploying these systems in real time means that we have to make them faster and more efficient. Because real-world data could be much more complex and unpredictable compared to the training data used in this study.

Directions for Further Research

- **Deep Learning Models**: Advanced deep learning techniques may be tried out in the future; these would include CNNs and RNNs. These models may have a way of detecting complex attack patterns that simpler models cannot.
- **Integration with Security Measures**: Other points of concentration would be the integration of those ML models with the existing cybersecurity tools, such as firewalls and intrusion detection systems, to provide a robust security system for EVSEs.

- **Scalability and Deployment**: Research should also investigate scalability regarding how these models can be deployed on larger networks of EVSE to ensure that they remain efficient even when the volume and complexity of the data grow.

By following these ideas, future studies can improve EVSE security and help support the growing use of EVs.

References

[1] F. Alanazi, “Electric Vehicles: Benefits, Challenges, and Potential Solutions for Widespread Adaptation,” *Appl. Sci.*, vol. 13, no. 10, p. 6016, May 2023, doi: 10.3390/app13106016.

[2] Government of Canada, “2030 Emissions Reduction Plan: Canada’s Next Steps for Clean Air and a Strong Economy.” Environment and Climate Change Canada, 2022. [Online]. Available: https://publications.gc.ca/collections/collection_2022/eccc/En4-460-2022-eng.pdf

[3] J. Johnson, T. Berg, B. Anderson, and B. Wright, “Review of Electric Vehicle Charger Cybersecurity Vulnerabilities, Potential Impacts, and Defenses,” *Energies*, vol. 15, no. 11, p. 3931, May 2022, doi: 10.3390/en15113931.

[4] E. D. Buedi, A. A. Ghorbani, S. Dadkhah, and R. L. Ferreira, “Enhancing EV Charging Station Security Using A Multi-dimensional Dataset : CICEVSE2024,” Mar. 11, 2024, *In Review*. doi: 10.21203/rs.3.rs-4046330/v1.

[5] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, “SMOTE: Synthetic Minority Over-sampling Technique,” *J. Artif. Intell. Res.*, vol. 16, pp. 321–357, Jun. 2002, doi: 10.1613/jair.953.

[6] SWASTIK, “SMOTE for Imbalanced Classification with Python,” Analytics Vidhya. Accessed: Nov. 27, 2024. [Online]. Available: https://www.analyticsvidhya.com/blog/2020/10/overcoming-class-imbalance-using-smote-techniques/

[7] B. Mahesh, “Machine Learning Algorithms - A Review,” *Int. J. Sci. Res. IJSR*, vol. 9, no. 1, pp. 381–386, Jan. 2020, doi: 10.21275/ART20203995.

[8] “Principal Component Analysis(PCA),” GeeksforGeeks. Accessed: Nov. 28, 2024. [Online]. Available: https://www.geeksforgeeks.org/principal-component-analysis-pca/

[9] “Machine Learning Tutorial,” GeeksforGeeks. Accessed: Nov. 25, 2024. [Online]. Available: https://www.geeksforgeeks.org/machine-learning/

[10] Z. J. Lee, T. Li, and S. H. Low, “ACN-Data: Analysis and Applications of an Open EV Charging Dataset,” in *Proceedings of the Tenth ACM International Conference on Future Energy Systems*, Phoenix AZ USA: ACM, Jun. 2019, pp. 139–149. doi: 10.1145/3307772.3328313.

[11] P. Inc., “Pecan Street Dataport.” Pecan Street Inc., Nov. 25, 2024. [Online]. Available: https://www.pecanstreet.org/dataport/

[12] T. Petersen, “Applied ML Training, Validation, and Test data set Cross validation.” [Online]. Available: https://www.nbi.dk/~petersen/Teaching/ML2022/Week2/ML2022_TrainValidTest_CrossValidation.pdf

[13] T. Shah, “About Train, Validation and Test Sets in Machine Learning | by Tarang Shah | Towards Data Science”.

[14] “Dataset for Classification,” GeeksforGeeks. Accessed: Nov. 25, 2024. [Online]. Available: https://www.geeksforgeeks.org/dataset-for-classification/

[15] E. Frank, M. A. Hall, and I. H. Witten, “Data Mining: Practical Machine Learning Tools and Techniques,” *Morgan Kaufmann*, 2016, [Online]. Available: https://ml.cms.waikato.ac.nz/weka

[16] G. M. Weiss, “Overview of DM with Weka Module”, [Online]. Available: https://storm.cis.fordham.edu/~gweiss/classes/cisc5790/slides/Weka-tutorial.pdf

[17] J. Phipps, “Reconnaissance in Cybersecurity: Types & Prevention,” eSecurity Planet. Accessed: Nov. 27, 2024. [Online]. Available: https://www.esecurityplanet.com/threats/how-hackers-use-reconnaissance/

[18] V. Chinnasamy, “What is SYN Attack and How to Prevent it? | Indusface Blog,” Indusface. Accessed: Nov. 27, 2024. [Online]. Available: https://www.indusface.com/blog/what-is-syn-synchronize-attack-how-the-attack-works-and-how-to-prevent-the-syn-attack/

[19] “What Is a UDP Flood DDoS Attack?,” Akamai. Accessed: Nov. 27, 2024. [Online]. Available: https://www.akamai.com/glossary/what-is-udp-flood-ddos-attack

[20] “What Is a Backdoor Attack? | CrowdStrike,” CrowdStrike.com. Accessed: Nov. 27, 2024. [Online]. Available: https://www.crowdstrike.com/en-us/cybersecurity-101/cyberattacks/backdoor-attack/

[21] C. Java, “Weka Data Preprocessing: Preparing Data for Machine Learning,” codefiner.com. Accessed: Nov. 24, 2024. [Online]. Available: https://codefiner.com/post/weka-data-preprocessing-preparing-data-for-machine-learning

[22] “RemoveUseless.” Accessed: Nov. 24, 2024. [Online]. Available: https://weka.sourceforge.io/doc.dev/weka/filters/unsupervised/attribute/RemoveUseless.html

[23] “Normalization in Machine Learning - Javatpoint,” www.javatpoint.com. Accessed: Nov. 27, 2024. [Online]. Available: https://www.javatpoint.com/normalization-in-machine-learning

[24] Sruthi, “Understanding Random Forest Algorithm With Examples,” Analytics Vidhya. Accessed: Nov. 28, 2024. [Online]. Available: https://www.analyticsvidhya.com/blog/2021/06/understanding-random-forest/

[25] “GBM in Machine Learning - Javatpoint,” www.javatpoint.com. Accessed: Nov. 28, 2024. [Online]. Available: https://www.javatpoint.com/gbm-in-machine-learning

[26] Anshul, “Guide on Support Vector Machine (SVM) Algorithm,” Analytics Vidhya. Accessed: Nov. 28, 2024. [Online]. Available: https://www.analyticsvidhya.com/blog/2021/10/support-vector-machinessvm-a-complete-guide-for-beginners/

[27] P. Cunningham and S. J. Delany, “k-Nearest Neighbour Classifiers - A Tutorial,” *ACM Comput. Surv.*, vol. 54, no. 6, pp. 1–25, Jul. 2022, doi: 10.1145/3459665.

[28] Anshul, “Logistic Regression: A Comprehensive Tutorial,” Analytics Vidhya. Accessed: Nov. 24, 2024. [Online]. Available: https://www.analyticsvidhya.com/blog/2021/08/conceptual-understanding-of-logistic-regression-for-data-science-beginners/

[29] sunil, “Naive Bayes Classifier Explained: Applications and Practical Problems,” Analytics Vidhya. Accessed: Nov. 24, 2024. [Online]. Available: https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/

[30] P. Huilgol, “Precision and Recall in Machine Learning,” Analytics Vidhya. Accessed: Nov. 24, 2024. [Online]. Available: https://www.analyticsvidhya.com/articles/precision-and-recall-in-machine-learning/

[31] “Confusion Matrix in Machine Learning,” Analytics Vidhya. Accessed: Nov. 24, 2024. [Online]. Available: https://www.analyticsvidhya.com/articles/confusion-matrix-in-machine-learning/