

Chapter 4

# **Evaluation and performance analysis of machine learning models for Identity and Access Management (IAM) attack detection**

Esther Chinwe Eze

*University of North Texas, United States*

## **4.0 Implementation**

This chapter documents how the experiment was implemented as well as a brief explanation of evaluation metrics.

### **4.1 Setup and Data Preparation**

To begin, all libraries required to carry out the experiment and prepare the data are imported as shown in the figure below.

```

1  # Import libraries
2  ---{r}
3  library(caret)
4  library(e4071)
5  library(randomForest)
6  library(ggplot2)
7  library(dplyr)
8  library(dp1yr)
9  set.seed(123)
10 ---

```

```

package 'caret' was built under R version 4.0.5>Loading required package: lattice
Loading required package: ggplot2
package 'ggplot2' was built under R version 4.0.5;use suppressPackageStartupMessages() to eliminate package startup messages
Registered S3 method overwritten by 'data.table':
  method
  from
  print.data.table
package 'e4071' was built under R version 4.0.5;package 'randomForest' was built under R version 4.0.5randomForest 4.6-14
Type ?PFnews() to see new Features/changes/bug fixes.
Attaching package: 'randomForest'

The following object is masked from 'package:ggplot2':
  margin

package 'dplyr' was built under R version 4.0.5
Attaching package: 'dplyr'

The following object is masked from 'package:randomForest':
  combine

The following objects are masked from 'package:stats':
  filter, lag

The following objects are masked from 'package:base':
  intersect, setdiff, setequal, union

```

**Figure 22: R Code: Importing Libraries**

### 4.1.1 Loading the Data into r and Preprocessing

Here, the data generated from the server in the form of logs have been transformed into two datasets called `ben_IAM` and `mal_IAM` which are then loaded/imported into the R studio IDE for exploration.

```

11 # Import datasets
12 ---{r}
13 #import datasets
14 ben_IAM <- read.csv("C:/Users/esthe/Desktop/MSC PROJECT/w502/ben_IAM.csv")
15 mal_IAM <- read.csv("C:/Users/esthe/Desktop/MSC PROJECT/w502/mal_IAM.csv")
16
17 tail(ben_IAM)
18 tail(mal_IAM)
19 ---

```

User	TimeStamp	overflow	TimestampDelta	Action	overflow2	overflow3	ActionDelta
996	Fen	44411	5757175926	logged in	579224537	9461342593	8882118056
997	Fen	44411	5757175926	0 Get-User-List	579224537	9461342593	8882118056
998	Fen	44411	5757175926	0 Get-Roles-of-User	5841782407	9461342593	3619560186
999	Fen	44411	5757175926	0 Get-User-Claim-Values	5841782407	9461342593	3619560186
1000	Fen	44411	5757175926	0 Get-User-List	5841782407	9461342593	3619560186
1001	Fen	44411	5757175926	0 Restart server	5841782407	9461342593	3619560186

**Figure 23: Loading Dataset**

Although the datasets were already cleaned, some basic cleaning like checking for missing values was done. There was no need to remove any feature as relevant features had already been selected during the process of data transformation. Handling missing values is as important as training because it could determine the accuracy of results. The mathematics underlying most models assumes that data is numeric and so should be free of missing values. Missing values in R codes could trigger errors while training.

Before checking for missing values, the two datasets loaded were joined together using the rbind() function. Afterward, missing values were checked using the sum() function in R. The result shows no missing value was found as presented in figure 18 below.

```

22 ~~~{r}
23 #bind dataset
24 #bind dataset
25 benma1_IAM <- rbind(ben_IAM, ma1_IAM)
26 str(benma1_IAM)
27 ~~~

'data.frame': 2002 obs. of 10 variables:
 $ User      : chr "Admin" "Admin" "Admin" "Admin" ...
 $ Timestamp : int 44388 44388 44388 44388 44388 44388 44388 44388 44388 44392 ...
 $ overflow  : num 0.935 0.935 0.936 0.936 0.936 ...
 $ TimestampDelta: num 0.00 0.00 8.45e-04 1.16e-05 2.31e-05 ...
 $ Action    : chr "Create" "Create" "Login" "Login" ...
 $ overflow2  : num 9.35e+09 9.35e+09 9.36e+09 9.36e+05 9.36e+09 ...
 $ overflow3  : num 9.35e+09 9.35e+09 9.35e+09 9.35e+09 9.35e+09 ...
 $ ActionDelta: num 0.00 0.00 8.45e+06 9.35e+09 8.80e+06 ...
 $ Password  : chr "rqntD16pd1B" "9hEkFp" "dpnysh" "r4rfjswtbcqc" ...
 $ outcome   : chr "Success" "Success" "Success" "Success" ...

28 ~~~{r}
29 #checking for missing
30 sum(complete.cases(benma1_IAM))
31 ~~~

[1] 2002

```

**Figure 24: Handling missing values**

### 4.1.2 Data Splitting

To split data, the CreateDataPartition () function in R was used as shown in the figure below. Before splitting, the classification label called ‘Outcome’ was converted to factor as data in numeric form while training. Data were split into the 80:20 ratio.

```

33 ~~~{r}
34 #changing training data to factor
35 benma1_IAMoutcome <- as.factor(benma1_IAMoutcome)
36 #splitting training and testing data
37 ~~~{r}
38 #splitting training and testing data
39 ~~~{r}
40 #splitting training and testing data
41 ~~~{r}
42 #splitting training and testing data
43 ~~~{r}
44 trainset <- createdatapartition(benma1_IAMoutcome, p=0.8, list = FALSE)
45 benma1_trainset <- benma1_IAM[trainset,]
46 benma1_testset <- benma1_IAM[-trainset,]
47 ~~~{r}
48 #view(benma1_trainset)
49 #view(benma1_testset)
50 ~~~{r}
51 ~~~{r}
52 ~~~{r}

```

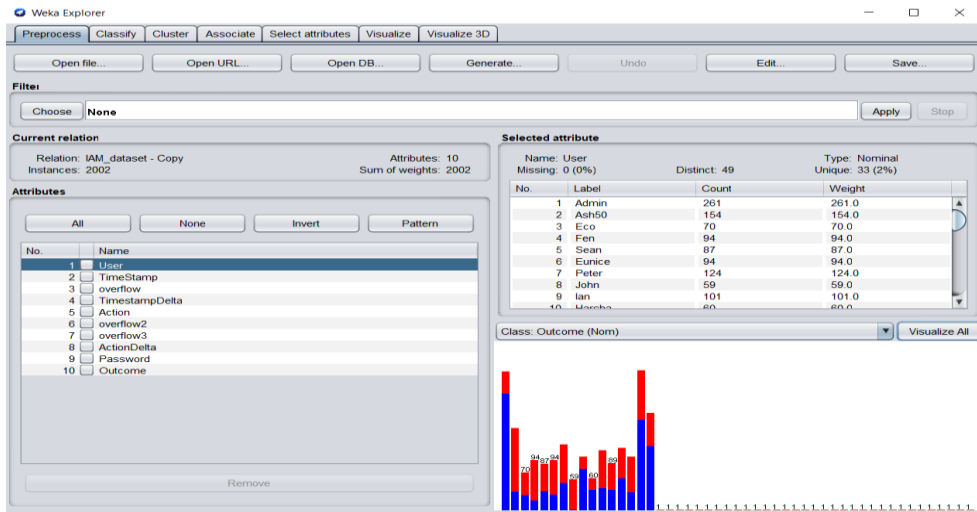
The Environment pane on the right shows the following objects:

- benma1\_IAM: 2002 obs. of 10 variables
- benma1\_testset: 400 obs. of 10 variables
- benma1\_trainset: 1602 obs. of 10 variables
- fit: Large knn3 (5 elements, 13.8 MB)
- ma1\_IAM: 1001 obs. of 10 variables
- na: List of 5
- rf: List of 19
- svmmodel: List of 30
- train\_set: int [1:1602, 1] 1 2 3 5 6 7 10 12 13
- 0: List of 9
- values: Factor w/ 2 levels "Failed","success"
- p1: Factor w/ 2 levels "Failed","success"
- p2: Factor w/ 2 levels "Failed","success"

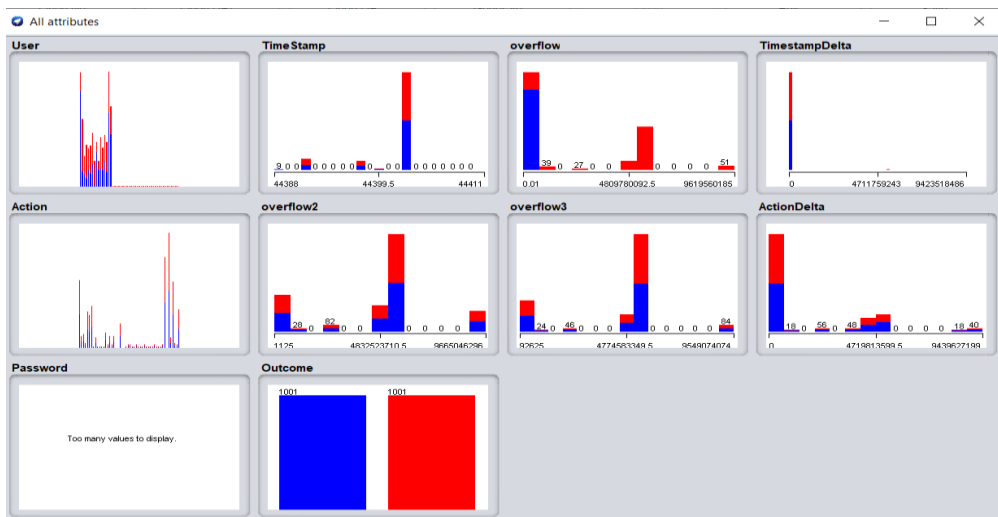
**Figure 25: Data Splitting**

### 4.1.3 Machine Learning Algorithms

During the process of training, the researcher encountered errors in the R codes and was unable to continue. The researcher intended to train with four (4) Machine Learning algorithms and select the one with the best performance. For this reason, the Weka GUI Machine Learning tool was selected as an alternative to continuing the experiment.



**Figure 26:** Loading of IAM\_dataset to Weka



**Figure 27:** Visualisation of All attributes

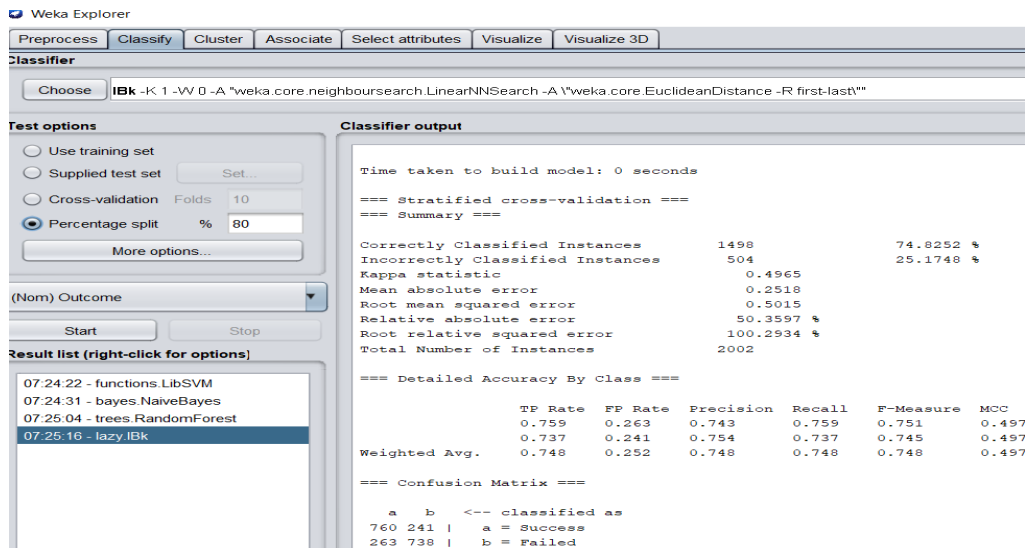


Figure 28: Selecting Algorithm for Testing

## 4.2 Evaluation Metrics: Confusion Matrix

In classification, many metrics are used for prediction and detection. Since the experiments for this project mainly distinguish normal activities from malicious activities, a confusion matrix will be used to determine the performance metrics. The confusion matrix is a table that describes in detail the results of the classification. The result from the confusion matrix be summated into four parts as shown below:

Confusion Matrix Component	Description
<b>True Positive (TP)</b>	This means that malicious instances are correctly classified to be malicious by the model.
<b>True Negative (TN)</b>	This means that benign instances are successfully identified to be benign by the model.
<b>False Positive (FP)</b>	This means that benign instances are wrongly classified to be malicious by the model.

---

**False Negative (FN)**

This means that malicious instances are wrongly identified as benign by the model.

---

**Table 4a:** Confusion Matrix Table

TRUE POSITIVE (TP)	FALSE NEGATIVE (FN)
FALSE POSITIVE (FP)	TRUE NEGATIVE (TN)

**Table 4b:** Confusion Matrix Table

The confusion matrix scenarios include classification indications such as accuracy, precision, recall, and F1-measure score (Porwal & Mukund, 2018). The result of the experiment follows

### 4.3 Performance of the Classifiers

Four (4) supervised Machine Learning algorithms were used for the experiment of this project – Random forest (RF), Naïve Bayes (NB), Support Vector Machine (SVM), and K- Nearest Neighbor (KNN). These classifiers were chosen because they are good for classification and are employed for their good performance. The evaluation was based on the accuracy of the classifiers, their F1-measure score, recall, and precision. Additional challenges encountered were having errors with the R code and for this reason, the researcher had to utilize a GUI-based ML tool called WEKA to help produce the results.

### 4.3.1 Random Forest Classifier (RF)

To determine the RF classifier’s performance, the model ran 100 iterations. Figure 29 below shows the result and performance of the RF classifier. The RF model had an accuracy of 100%.

The screenshot shows the Weka Classifier interface. The 'Classifier' dropdown is set to 'RandomForest -P 100 -I 100 -num-slots 1 -K 0 -M 1.0 -V 0.001 -S 1'. Under 'Test options', 'Percentage split' is selected at 80%. The 'Result list' shows '04:07:22 - trees.RandomForest' as the selected classifier. The 'Classifier output' pane displays the following text:

```

=== Run information ===

Scheme:      weka.classifiers.trees.RandomForest -P 100 -I 100 -num-slots 1 -K 0 -M 1.0 -V 0.001
Relation:    IAM_dataset
Instances:   2002
Attributes:  10
             User
             Timestamp
             overflow
             TimestampDelta
             Action
             overflow2
             overflow3
             ActionDelta
             Password
             Outcome

Test mode:   split 80.0% train, remainder test

=== Classifier model (full training set) ===

RandomForest

Bagging with 100 iterations and base learner

weka.classifiers.trees.RandomTree -K 0 -M 1.0 -V 0.001 -S 1 --do-not-check-capabilities

Time taken to build model: 0.66 seconds

=== Evaluation on test split ===

=== Summary ===

Correctly Classified Instances      2002           100      %
Incorrectly Classified Instances     0              0      %
Kappa statistic                    1
Mean absolute error                 0.2528
Root mean squared error             0.2569
Relative absolute error             50.5681 %
Root relative squared error        51.3859 %
Total Number of Instances          2002

=== Detailed Accuracy By Class ===

              TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Cl
              1.000    0.000    1.000     1.000    1.000     1.000    1.000    1.000    Su
              1.000    0.000    1.000     1.000    1.000     1.000    1.000    1.000    Fa
Weighted Avg.  1.000    0.000    1.000     1.000    1.000     1.000    1.000    1.000

=== Confusion Matrix ===

  a  b  <-- classified as
1001  0 | a = Success
  0 1001 | b = Failed
  
```

**Figure 29:** Performance Result of RF Classifier

Figure 30 below shows the confusion matrix of the classifier. The metrics for the evaluation have already been explained in Table 4.

**True Positives (TP):** The Random Forest model correctly classified 1001 malicious instances to be malicious.

**True Negatives (TN):** The model correctly classified 1001 benign instances to be benign.

No False Positive (FP) and False Negative(FN).

1001	0
0	1001

**Figure 30:** Confusion Matrix showing the performance of the RF Classifier

### 4.3.2 Naïve Bayes Classifier (NB)

The NB classifier measured an accuracy of 91%.

The screenshot shows the Weka Classifier window. The classifier selected is 'RandomForest-P 100-I 100-num-slots 1-K 0-M 1.0-V 0.001-S 1'. Under 'Test options', 'Percentage split' is set to 80%. The 'Classifier output' pane shows the following information:

```

=== Run information ===

Scheme:      weka.classifiers.bayes.NaiveBayes
Relation:    IAM_dataset
Instances:   2002
Attributes:  10
  User
  Timestamp
  overflow
  TimestampDelta
  Action
  overflow2
  overflow3
  ActionDelta
  Password
  Outcome

Test mode:   split 80.0% train, remainder test

=== Classifier model (full training set) ===

Naive Bayes Classifier
  
```

The 'Result list' on the left shows several entries, with the most recent being '04:00:30 - bayes.NaiveBayes'.



```

Time taken to build model: 0.02 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances      1836           91.7083 %
Incorrectly Classified Instances    166           8.2917 %
Kappa statistic                    0.8342
Mean absolute error                 0.083
Root mean squared error             0.2879
Relative absolute error             16.5914 %
Root relative squared error         57.5871 %
Total Number of Instances          2002

=== Detailed Accuracy By Class ===

          TP Rate  FP Rate  Precision  Recall  F-Measure  MCC      ROC Area  PRC Area  Cl
1001      1.000    0.166    0.858     1.000   0.923     0.846    0.883    0.761    Su
0         0.834    0.000    1.000     0.834   0.910     0.846    0.883    0.932    Fa
Weighted Avg.    0.917    0.083    0.929     0.917   0.917     0.846    0.883    0.847

=== Confusion Matrix ===

  a    b  <-- classified as
1001  0 |  a = Success
166  835 |  b = Failed

```

**Figure 31:** Performance result of Naïve Bayes Classifier

**True Positives (TP):** The NB classifier model correctly identified 1001 True Positives, this indicates that 1001 successful (malicious) instances were predicted correctly.

**True Negatives (TN):** As observed in the performance result of Figure 31, the number of true negatives obtained is 835 which indicates the number of benign instances correctly predicted.

**False Positives (FP):** The model identified 166 false positives which means that 166 benign instances were wrongly classified as malicious. The false positives are also known as type 1 errors. This type of error in a real-world situation may not appear critical but in the long run, it may lead to losses while attempting to resolve what does not happen.

**False Negatives (FN):** As shown in the performance result, 0 false negatives were identified which indicates that no malicious instances were wrongly predicted as benign categories. False negatives are also known as type 2 errors, the implication of this type of error in an organization could result in serious damage.

1001	0
166	835

**Figure 32:** Confusion Matrix showing the Naive Bayes Classifier

### 4.3.3 Support Vector Machine (SVM)

The SVM model had an accuracy of 88%.

The screenshot shows the Weka Explorer interface with the LibSVM classifier selected. The 'Test options' section shows 'Percentage split' at 80%. The 'Classifier output' pane displays the following information:

```

=== Run information ===

Scheme:      weka.classifiers.functions.LibSVM -S 0 -K 2 -D 3 -G 0.0 -R 0.0 -N 0.5 -M 40.0 -C 1.0 -E 0.001 -P 0.1 -model "C:\Program Files\Weka-3-9-5" -seed 1
Relation:    IAM_dataset
Instances:   2002
Attributes:  10
User
TimeStamp
overflow
TimestampDelta
Action
overflow2
overflow3
ActionDelta
Password
Outcome

Test mode:   split 80.0% train, remainder test

=== Classifier model (full training set) ===

LibSVM wrapper, original code by Yasser EL-Manzalawy (= WLSVM)

Time taken to build model: 0.29 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 0.03 seconds

=== Summary ===

```

The 'Result list' shows the SVM model as the top performer:

- 04:00:00 - functions.LibSVM

The detailed performance results are as follows:

```

Time taken to build model: 0.14 seconds

== Stratified cross-validation ==
== Summary ==

Correctly Classified Instances      1767           88.2617 %
Incorrectly Classified Instances    235           11.7383 %
Kappa statistic                    0.7652
Mean absolute error                 0.1174
Root mean squared error             0.3426
Relative absolute error             23.4765 %
Root relative squared error         68.5223 %
Total Number of Instances          2002

== Detailed Accuracy By Class ==

```

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Clas.
Weighted Avg.	0.883	0.117	0.905	0.883	0.881	0.787	0.883	0.846	

```

== Confusion Matrix ==

 a   b  <-- classified as
766 235 | a = Success
  0 1001 | b = Failed

```

Figure 33: Performance result of SVM Classifier

**True Positives (TP):** The model identified 766 true positives which means that 766 malicious instances were correctly predicted as malicious.

**True Negatives (TN):** As observed in the performance result in Figure 33, the number of true negatives obtained is 1001 which indicates the number of benign instances correctly predicted.

**False Positives (FP):** The model identified 0 benign instances that were wrongly classified as malicious categories.

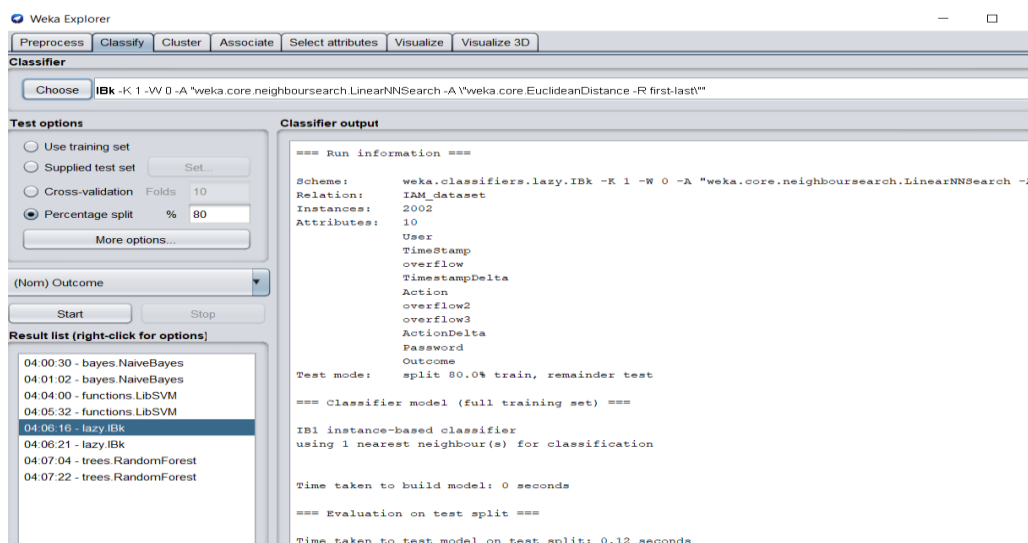
**False Negatives (FN):** The model identified 235 false negatives which means 235 malicious instances were wrongly classified as benign instances.

766	235
0	1001

**Figure 34:** Confusion Matrix of SVM Classifier

### 4.3.4 K-NEAREST NEIGHBOR (KNN)

KNN had an accuracy of 74% which is the lowest of all 4 algorithm



```

=== Summary ===

Correctly Classified Instances      1498           74.8252 %
Incorrectly Classified Instances     504           25.1748 %
Kappa statistic                     0.4965
Mean absolute error                  0.2518
Root mean squared error              0.5015
Relative absolute error              50.3597 %
Root relative squared error          100.2934 %
Total Number of Instances           2002

=== Detailed Accuracy By Class ===

                TP Rate  FP Rate  Precision  Recall   F-Measure  MCC       ROC Area  PRC Area  C.
                0.759   0.263   0.743     0.759   0.751     0.497    0.868    0.854    S.
                0.737   0.241   0.754     0.737   0.745     0.497    0.868    0.851    F.
Weighted Avg.   0.748   0.252   0.748     0.748   0.748     0.497    0.868    0.853

=== Confusion Matrix ===

  a  b  <-- classified as
760 241 |  a = Success
263 738 |  b = Failed

```

**Figure 35:** Performance result of KNN Classifier

**True Positives (TP):** The KNN classifier correctly identified 760 True Positives, indicating the correct prediction of 760 malicious instances.

**True Negatives (TN):** As observed in the performance result of Figure 35, the number of true negatives obtained is 738 which indicates the number of benign instances correctly predicted.

**False Positives (FP):** The model identified 263 benign categories that were wrongly classified as malicious instances.

**False Negatives (FN):** As shown in the performance result, 241 false negatives were identified which indicates that 241 malicious categories were wrongly predicted as benign categories.

760	241
263	738

**Figure 36:** Confusion Matrix for KNN

#### 4.4 Comparison of evaluation metrics

As shown in Table 5 below, four (4) supervised Machine Learning algorithms were experimented on to test the performance of each algorithm. The evaluation metrics included the Accuracy, Precision, Recall, and F1-measure scores of each model. As seen in almost all cases, the Random Forest (RF) algorithm gave better results as compared to other algorithms. The KNN records the lowest result across the four metrics. Based on the performance result, the Random Forest algorithm was selected for the project. This proves that In terms of IAM attack detection, the Random Forest algorithm has great potential to make the IAM process more secure, efficient, robust, and resilient in dealing with IAM attacks. Research shows that it has high performance for detection especially in areas of malicious detection. From the experiment, the result shows that Random Forest was able to correctly predict normal activities from malicious activities.

Model	Accuracy	Precision	Recall	F1-Measure
Random Forest	1	1.00	1.00	1.00
Naïve Baye	0.91	0.92	0.92	0.92
SVM	0.88	0.90	0.88	0.88
KNN	0.74	0.74	0.74	0.74

**Table 5:** Accuracy, Precision, Recall & F1-Measure Metrics

#### 4.5 Result with compliance to project requirements and objectives

This sub-section compares the result achieved to the project's functional requirement to evaluate what has been achieved and what has not.

**Table 6:** Result comparison with Project Functional Requirement

<b>S/N</b>	<b>REQUIREMENT DECLARATION</b>	<b>PRIORITIZATION</b>	<b>COMMENT</b>	<b>PASS/FAIL</b>
1	The model must be able to detect IAM attacks.	Must	If properly implemented, the selected model will help detect attacks.	Pass
2	The model should be able to detect attacks based on the input dataset.	Should	The experiment shows that Random Forest was able to correctly predict normal activities from malicious activities based on the input dataset.	Pass
3	Achieve an accuracy above 75% in the testing phase	Should	The result from the experiment shows Random Forest with the highest accuracy of 100%.	Pass

**Table 7:** Non-Functional requirement

S/N O	REQUIREMEN T	DESCRIPTIO N	PRIORI TIZATI ON	COMMENT	PASS/FA IL
1	<b>Reproducible</b>	The result and code should be reproducible and accessible for reproducing the result. Therefore, the code will be written in R programming to ensure that it can be reusable.	Should	During the implementation stage, the R codes developed some errors and remained unsolved. The researcher opted to use an alternative to save time. The alternative tool used was WEKA (a GUI-based ML tool). However, the result is still reproducible.	Pass
2	<b>Adaptability</b>	The proposed detection model should be able to adjust to modifications in terms of features.	Could	The chosen algorithm is flexible and open for future improvement for optimum performance. Modifications can also be made to the	Pass

---

				dataset features.	
<b>3</b>	<b>Security Necessities</b>	The model should be able to detect unauthorized access	Must	The aspect of user authentication was focused on and the result showed that the algorithm was able to predict normal activities from malicious activities.	Pass
<b>4</b>	<b>Compatibility</b>	It should be compatible with either desktop or laptop Windows operating system.	should	Model is compatible	Pass
<b>5</b>	<b>Performance</b>	Should work as expected.	Must	Upon implementation, the model will work as intended i.e for the detection	Pass
<b>6</b>	<b>Efficiency</b>	The output is required to be more accurate and should have a low false-positives (FP)	Should	This was achieved as the RF algorithm had no false positives or	Pass

---



---

rate and false negatives.	false negatives.
---------------------------	------------------

---

**Table 8:** Evaluation of Aim & Objectives

---

OBJECTIVES	COMMENT
Review various types of attacks targeted against IAM.	Achieved in the investigation report.
Review existing Machine Learning (ML) approaches, techniques, and tools in IAM attack detection.	Achieved in the investigation report.
Collect data by Setting up a testbed that mimics normal and malicious activities.	Achieved
Use data to train machine learning algorithms that distinguish normal activities from malicious activities.	Achieved
Select the best Machine Learning (ML) algorithm that predicts normal from malicious activities.	Achieved

---