

**LABORATORY PROJECT**

**(BITE393J)**

# EFFECTIVE IDS CLASSIFICATION FOR IOT

**Submitted by:**

**ARYAN (21BIT0565)**

**PROJECT GUIDE:**

**Dr. Chandra Segar. T**

**Abstract :**

In the interconnected world of the Internet of Things (IoT), the integrity and security of devices and networks are pivotal. Intrusion Detection Systems (IDS) serve as the frontline defense, detecting and thwarting malicious activities within IoT ecosystems. However, the dynamic and diverse nature of IoT data poses significant challenges to traditional intrusion detection methods, necessitating the adoption of advanced machine learning models.

This research endeavours to evaluate the efficacy of prominent machine learning algorithms, including Random Forest, KNN, SVC, XG Boost, Decision Tree, and Linear Regression, in effectively classifying IDS data tailored for IoT environments. This study aims to ascertain the capabilities of these models in accurately discerning intrusions amidst the complexities of IoT-generated data.

**Introduction:**

The way we interact with technology has changed dramatically as a result of the Internet of Things' (IoT) widespread adoption. Smart device ecosystems are now interconnected, blending the digital and physical worlds. But this connectivity also brings with it an unparalleled level of risk, as bad actors looking to take advantage of security flaws target IoT systems as their primary targets. Within this framework, intrusion detection systems (IDS) become like indispensible sentinels, charged with the vital responsibility of keeping an eye on network activity and spotting unusual activity suggestive of possible breaches.

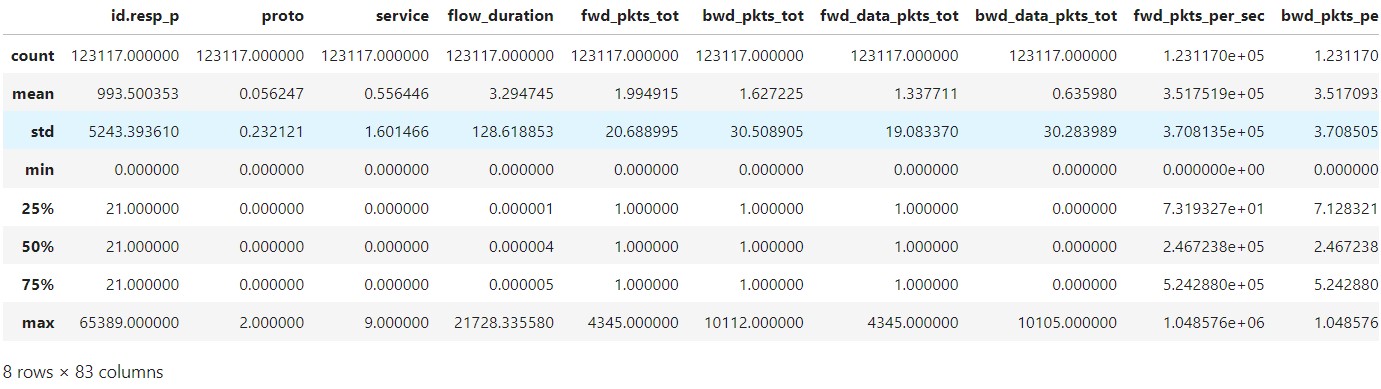
Although typical network environments have shown that classic intrusion detection systems (IDS) function well, the distinctive features of IoT ecosystems provide new obstacles. More complex and adaptable intrusion detection systems are required due to the sheer amount, heterogeneity, and real-time nature of Internet of Things data. This is where the importance of utilizing

**Methodologies :**

# Load libraries import pandas as pd import numpy as np import xgboost as xgb from sklearn.datasets import make\_classification from sklearn.ensemble import RandomForestClassifier from sklearn.model\_selection import train\_test\_split from pandas.plotting import scatter\_matrix import matplotlib.pyplot as plt from sklearn import model\_selection from sklearn import metrics from sklearn.metrics import classification\_report from sklearn.metrics import confusion\_matrix from sklearn.metrics import accuracy\_score from sklearn.ensemble import RandomForestClassifier from sklearn.tree import DecisionTreeClassifier import matplotlib.pyplot as plt import seaborn as sns

data = pd.read\_csv('E:/project/dd.csv') #reading csv file

data.describe()



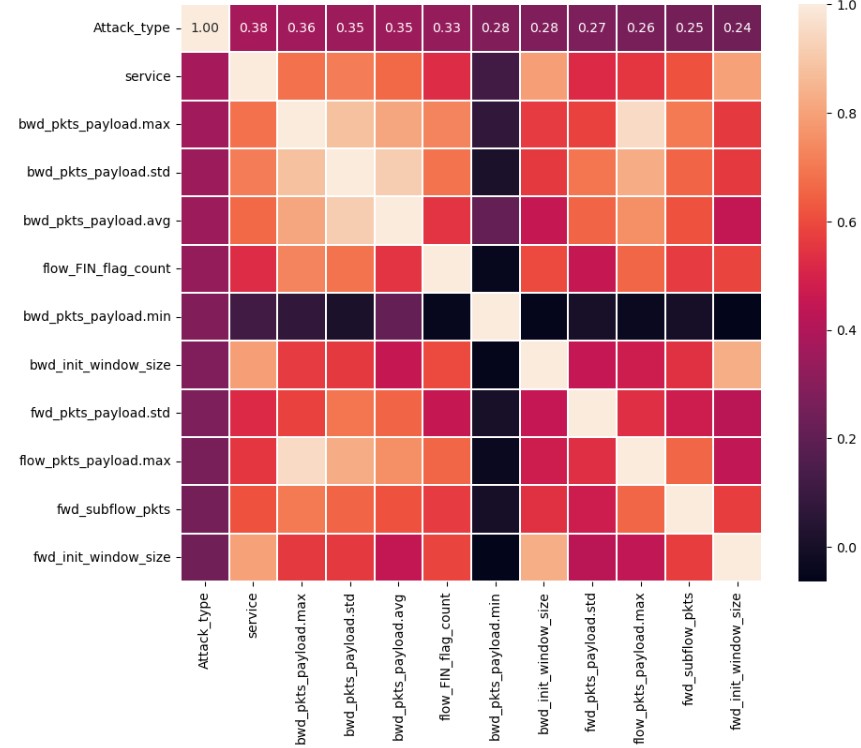
# Split-out validation dataset array = data.values X = array[:,0:82]

y = array[:,82]

# Select the top 12 most correlated features top\_corr\_features = data.corr().nlargest(12, 'Attack\_type')['Attack\_type'].index correlation\_matrix = data[top\_corr\_features].corr()

# Plot the correlation heatmap plt.figure(figsize=(10, 8))

sns.heatmap(correlation\_matrix, annot=True, fmt=".2f", linewidths=1.2) plt.show()



feature\_cols = ['id.resp\_p', 'proto', 'service', 'flow\_duration',

'fwd\_pkts\_tot', 'bwd\_pkts\_tot', 'fwd\_data\_pkts\_tot',

'bwd\_data\_pkts\_tot', 'fwd\_pkts\_per\_sec', 'bwd\_pkts\_per\_sec',

'flow\_pkts\_per\_sec', 'down\_up\_ratio', 'fwd\_header\_size\_tot',

'fwd\_header\_size\_min', 'fwd\_header\_size\_max',

'bwd\_header\_size\_tot', 'bwd\_header\_size\_min', 'bwd\_header\_size\_max',

'flow\_FIN\_flag\_count', 'flow\_SYN\_flag\_count',

'flow\_RST\_flag\_count', 'flow\_PSH\_flag\_count', 'bwd\_PSH\_flag\_count',

'flow\_ACK\_flag\_count', 'flow\_URG\_flag\_count', 'bwd\_URG\_flag\_count',

'flow\_CWR\_flag\_count', 'flow\_ECE\_flag\_count',

'fwd\_pkts\_payload.min', 'fwd\_pkts\_payload.max',

'fwd\_pkts\_payload.tot', 'fwd\_pkts\_payload.avg', 'fwd\_pkts\_payload.std', 'bwd\_pkts\_payload.min', 'bwd\_pkts\_payload.max',

'bwd\_pkts\_payload.tot', 'bwd\_pkts\_payload.avg', 'bwd\_pkts\_payload.std',

'flow\_pkts\_payload.min', 'flow\_pkts\_payload.max',

'flow\_pkts\_payload.tot', 'flow\_pkts\_payload.avg', 'flow\_pkts\_payload.std',

'fwd\_iat.min', 'fwd\_iat.max', 'fwd\_iat.tot', 'fwd\_iat.avg',

'fwd\_iat.std', 'bwd\_iat.min', 'bwd\_iat.max', 'bwd\_iat.tot',

'bwd\_iat.avg', 'bwd\_iat.std', 'flow\_iat.min', 'flow\_iat.max',

'flow\_iat.tot', 'flow\_iat.avg', 'flow\_iat.std',

'payload\_bytes\_per\_second', 'fwd\_subflow\_pkts', 'bwd\_subflow\_pkts',

'fwd\_subflow\_bytes', 'bwd\_subflow\_bytes',

'fwd\_bulk\_bytes', 'bwd\_bulk\_bytes', 'fwd\_bulk\_packets',

'bwd\_bulk\_packets', 'fwd\_bulk\_rate', 'bwd\_bulk\_rate',

'active.min', 'active.max', 'active.tot', 'active.avg',

'active.std', 'idle.min', 'idle.max', 'idle.tot',

'idle.avg', 'idle.std', 'fwd\_init\_window\_size',

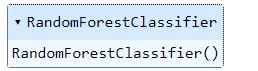
'bwd\_init\_window\_size', 'fwd\_last\_window\_size', 'Attack\_type']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3)

#Create a Gaussian Classifier

clf=RandomForestClassifier(n\_estimators=100)

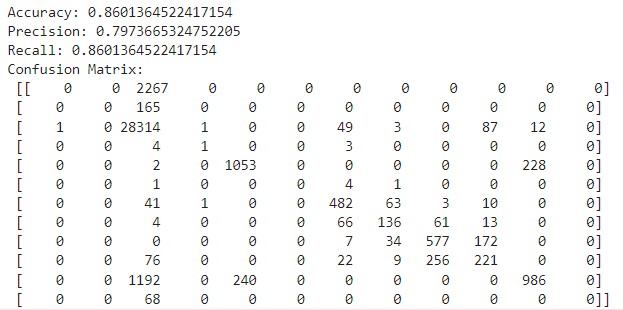
#Train the model using the training sets y\_pred=clf.predict(X\_test) clf.fit(X\_train,y\_train)



y\_pred=clf.predict(X\_test)

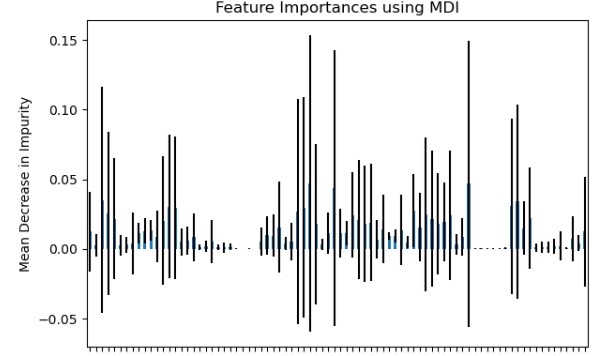
|  |
| --- |
| from sklearn.ensemble import RandomForestClassifier from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, confusion\_matrix    # Instantiate the Random Forest classifier random\_forest = RandomForestClassifier(n\_jobs=-1, random\_state=15)  # Train the Random Forest classifier using training data random\_forest.fit(X\_train, y\_train)    # Evaluation y\_pred\_test = random\_forest.predict(X\_test)  accuracy = accuracy\_score(y\_true=y\_test, y\_pred=y\_pred\_test) precision = precision\_score(y\_true=y\_test, y\_pred=y\_pred\_test, average='weighted') # specify average parameter recall = recall\_score(y\_true=y\_test, y\_pred=y\_pred\_test, average='weighted') # specify average parameter    # Confusion Matrix  CM = confusion\_matrix(y\_true=y\_test, y\_pred=y\_pred\_test) print("Accuracy:", accuracy) print("Precision:", precision) |

print("Recall:", recall) print("Confusion Matrix:\n", CM)



RandomForestClassifier(bootstrap=True, class\_weight=None, criterion='gini', max\_depth=None, max\_features='auto', max\_leaf\_nodes=None, min\_impurity\_decrease=0.0, min\_samples\_leaf=1, min\_samples\_split=2, min\_weight\_fraction\_leaf=0.0, n\_estimators=100, n\_jobs=1, oob\_score=False, random\_state=None, verbose=0, warm\_start=False)

|  |
| --- |
| # Compute feature importances importances = clf.feature\_importances\_ std = np.std([tree.feature\_importances\_ for tree in clf.estimators\_], axis=0)  # Plot the impurity-based importance  feature\_names = [f"feature {i}" for i in range(X.shape[1])] forest\_importances = pd.Series(importances, index=feature\_names)  fig, ax = plt.subplots() forest\_importances.plot.bar(yerr=std, ax=ax) ax.set\_title("Feature Importances using MDI") ax.set\_ylabel("Mean Decrease in Impurity") fig.tight\_layout() plt.show() |

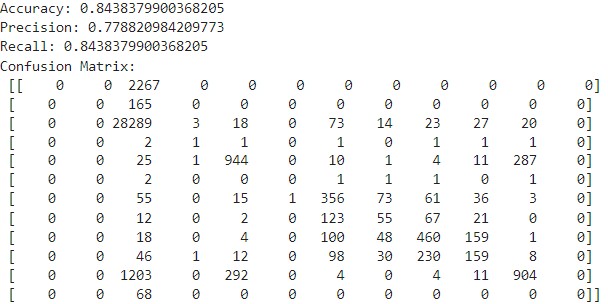


|  |
| --- |
| #KNN MODEL from sklearn.neighbors import KNeighborsClassifier  knn\_model = KNeighborsClassifier(n\_neighbors=5)  # Train the model knn\_model.fit(X\_train, y\_train) # Make predictions on the test set y\_pred = knn\_model.predict(X\_test) |

|  |
| --- |
| from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, confusion\_matrix    # Instantiate the KNeighborsClassifier knn\_classifier = KNeighborsClassifier()  # Train the KNeighborsClassifier using training data knn\_classifier.fit(X\_train, y\_train)    # Evaluation  y\_pred\_test = knn\_classifier.predict(X\_test)  accuracy = accuracy\_score(y\_true=y\_test, y\_pred=y\_pred\_test) precision = precision\_score(y\_true=y\_test, y\_pred=y\_pred\_test, average='weighted') # specify average parameter  recall = recall\_score(y\_true=y\_test, y\_pred=y\_pred\_test, average='weighted') # specify average parameter    # Confusion Matrix  CM = confusion\_matrix(y\_true=y\_test, y\_pred=y\_pred\_test) print("Accuracy:", accuracy) |

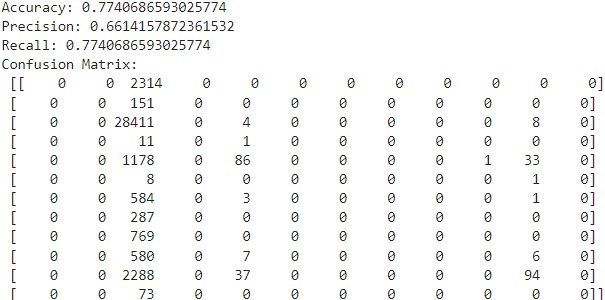
print("Precision:", precision) print("Recall:", recall)

print("Confusion Matrix:\n", CM)



|  |
| --- |
| # SVC model from sklearn.svm import SVC svc\_model = SVC(random\_state=42)  # Train the model svc\_model.fit(X\_train, y\_train)    # Make predictions on the test set y\_pred = svc\_model.predict(X\_test) |

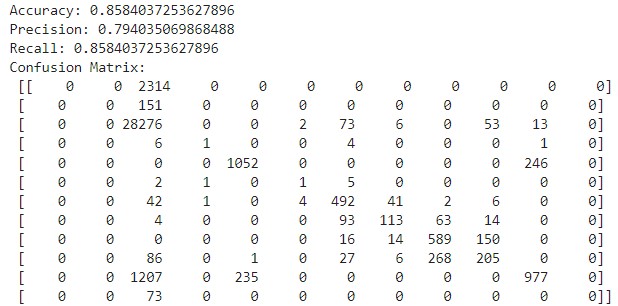
|  |
| --- |
| from sklearn.svm import SVC from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, confusion\_matrix    # Instantiate the Support Vector Classifier svc\_classifier = SVC()    # Train the SVC classifier using training data svc\_classifier.fit(X\_train, y\_train)    # Evaluation  y\_pred\_test = svc\_classifier.predict(X\_test)  accuracy = accuracy\_score(y\_true=y\_test, y\_pred=y\_pred\_test) precision = precision\_score(y\_true=y\_test, y\_pred=y\_pred\_test, average='weighted') # specify average parameter |
| recall = recall\_score(y\_true=y\_test, y\_pred=y\_pred\_test, average='weighted') # specify average parameter    # Confusion Matrix  CM = confusion\_matrix(y\_true=y\_test, y\_pred=y\_pred\_test) print("Accuracy:", accuracy) print("Precision:", precision) print("Recall:", recall) print("Confusion Matrix:\n", CM) |



|  |
| --- |
| import xgboost as xgb  # Convert the float labels to integers y\_train = y\_train.astype(int)    # XGBoost model  xgb\_model = xgb.XGBClassifier(random\_state=42)    # Train the model xgb\_model.fit(X\_train, y\_train)    # Make predictions on the test set y\_pred = xgb\_model.predict(X\_test) |

|  |
| --- |
| from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, confusion\_matrix    # Instantiate the XGBoost classifier xgb\_classifier = xgb.XGBClassifier()  # Train the XGBoost classifier using training data xgb\_classifier.fit(X\_train, y\_train) |

|  |
| --- |
| # Evaluation y\_pred\_test = xgb\_classifier.predict(X\_test) accuracy = accuracy\_score(y\_true=y\_test, y\_pred=y\_pred\_test) precision = precision\_score(y\_true=y\_test, y\_pred=y\_pred\_test, average='weighted') # specify average parameter recall = recall\_score(y\_true=y\_test, y\_pred=y\_pred\_test, average='weighted') # specify average parameter    # Confusion Matrix  CM = confusion\_matrix(y\_true=y\_test, y\_pred=y\_pred\_test) print("Accuracy:", accuracy) print("Precision:", precision) print("Recall:", recall) print("Confusion Matrix:\n", CM) |

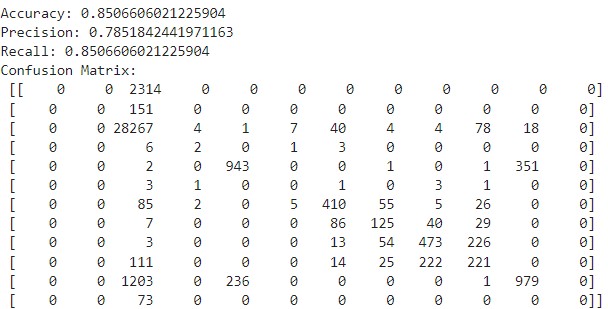


|  |
| --- |
| #LINEAR REGRESSION  from sklearn.linear\_model import LinearRegression from sklearn.metrics import mean\_squared\_error, r2\_score  # Linear Regression model  linear\_model = LinearRegression()    # Train the model  linear\_model.fit(X\_train, y\_train)  # Make predictions on the test set y\_pred = linear\_model.predict(X\_test)    # Evaluate the model |
| mse = mean\_squared\_error(y\_test, y\_pred) r2 = r2\_score(y\_test, y\_pred)  print("Mean Squared Error for Linear Regression:", mse) print("R-squared (Coefficient of Determination) for Linear Regression:", r2) |



|  |
| --- |
| from sklearn.tree import DecisionTreeClassifier  # Decision Tree model tree\_model = DecisionTreeClassifier(random\_state=42)  # Train the model tree\_model.fit(X\_train, y\_train)    # Make predictions on the test set y\_pred = tree\_model.predict(X\_test) |

|  |
| --- |
| from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, confusion\_matrix    # Instantiate the Decision Tree classifier decision\_tree = DecisionTreeClassifier(random\_state=15)  # Train the Decision Tree classifier using training data decision\_tree.fit(X\_train, y\_train)    # Evaluation y\_pred\_test = decision\_tree.predict(X\_test) accuracy = accuracy\_score(y\_true=y\_test, y\_pred=y\_pred\_test) precision = precision\_score(y\_true=y\_test, y\_pred=y\_pred\_test, average='weighted') # specify average parameter  recall = recall\_score(y\_true=y\_test, y\_pred=y\_pred\_test, average='weighted') # specify average parameter    # Confusion Matrix  CM = confusion\_matrix(y\_true=y\_test, y\_pred=y\_pred\_test) print("Accuracy:", accuracy) print("Precision:", precision) print("Recall:", recall) print("Confusion Matrix:\n", CM) |



**Results :**

**RAW DATA :**

|  |  |  |
| --- | --- | --- |
| **Random Forest** |  | **Logistic Regression** |
| **Accuracy : 0.71257** | **Accuracy : 0.5601** |
| **Error Rate : 0.28474** | **Error Rate : 0.43989** |
| **Precision : 0.85628** | **Precision : 0.78005** |
| **Sensitivity : 0.85628** | **Sensitivity : 0.78005** |
| **Specificity : error** | **Specificity : error** |
| **F1 score : 0.42814** | **F1 score : 0.39002** |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | | **Naïve Bayes** | | **Accuracy : 0.31863** | | **Error Rate : 0.68136** | | **Precision : 0.06919** | | **Sensitivity : 0.06919** | | **Specificity : 0.5** | | **F1 score : 0.0346** | | |  | | --- | | **KNN Model** | | **Accuracy : 0.67679** | | **Error Rate : 0.3232** | | **Precision : 0.83839** | | **Sensitivity : 0.83839** | | **Specificity : error** | | **F1 score : 0.4192** | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | | **SVC Model** | | **Accuracy : 0.54526** | | **Error Rate : 0.45473** | | **Precision : 0.77263** | | **Sensitivity : 0.77263** | | **Specificity : error** | | **F1 score : 0.38632** | | |  | | --- | | **XG Boost** | | **Accuracy : 0.71193** | | **Error Rate : 0.28806** | | **Precision : 0.85596** | | **Sensitivity : 0.85596** | | **Specificity : error** | | **F1 score : 0.42798** | |

|  |
| --- |
| **Decision Tree** |
| **77Accuracy : 0.69765** |
| **Error Rate : 0.30234** |
| **Precision : 0.84881** |
| **Sensitivity : 0.84881** |
| **Specificity : error** |
| **F1 score : 0.42441** |

**REFINED DATA :**

|  |  |  |
| --- | --- | --- |
| **Random Forest** |  | **Logistic Regression** |
| **Accuracy : 0.9994** | **Accuracy : 0.86917** |
| **Error Rate : 0.00059** | **Error Rate : 0.13082** |
| **Precision : 0.9997** | **Precision : 0.93458** |
| **Sensitivity : 0.9997** | **Sensitivity : 0.93458** |
| **Specificity : error** | **Specificity : error** |
| **F1 score : 0.49985** | **F1 score : 0.46729** |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | | **Naïve Bayes** | | **Accuracy : 0.64349** | | **Error Rate : 0.13082** | | **Precision : 0.17825** | | **Sensitivity : 0.17825** | | **Specificity : error** | | **F1 score : 0.08912** | | | | |  | | --- | | **KNN Model** | | **Accuracy : 0.97639** | | **Error Rate : 0.0236** | | **Precision : 0.98819** | | **Sensitivity : 0.98819** | | **Specificity : error** | | **F1 score : 0.49409** | | |
| **SVC Model** |  | | **XG Boost** |
| **Accuracy : 0.54857** | **Accuracy : 1** |
| **Error Rate : 0.45142** | **Error Rate : 0** |
| **Precision : 0.77428** | **Precision : 1** |
| **Sensitivity : 0.77428** | **Sensitivity : 1** |
| **Specificity : error** | **Specificity : 0** |
| **F1 score : 0.38714** | **F1 score : 0.5** |

|  |
| --- |
| **Decision Tree** |
| **Accuracy : 1** |
| **Error Rate : 0** |
| **Precision : 1** |
| **Sensitivity : 1** |
| **Specificity : 0** |
| **F1 score : 0.5** |

**10x4 Validation**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Models** |  | **Repetition 1** | **Repetition 2** | **Repetition 3** | **Repetition 4** |
| **Random Forest** | **Mean**  **Accuracy** | 0.85 | 0.85 | 0.85 | 0.85 |
|  | **STD Deviation** | 0 | 0 | 0 | 0 |
| **Logistic**  **Regression** | **Mean**  **Accuracy** | 0.77 | 0.77 | 0.77 | 0.77 |
|  | **STD Deviation** | 0 | 0 | 0 | 0 |
| **Naïve Bayes** | **Mean**  **Accuracy** | 0.18 | 0.18 | 0.18 | 0.18 |
|  | **STD Deviation** | 0 | 0 | 0 | 0 |
| **KNN Model** | **Mean**  **Accuracy** | 0.16 | 0.16 | 0.16 | 0.16 |
|  | **STD Deviation** | 0 | 0 | 0 | 0 |
| **XG Boost** | **Mean**  **Accuracy** | 0.85 | 0.85 | 0.85 | 0.85 |
|  | **STD Deviation** | 0 | 0 | 0 | 0 |
| **Decision Tree** | **Mean**  **Accuracy** | 0.85 | 0.85 | 0.85 | 0.85 |
|  | **STD Deviation** | 0 | 0 | 0 | 0 |