Configuration Manual

MSc Research Project
MSc Cloud Computing

Gauri Misra
Student ID: x20259611

School of Computing
National College of Ireland

Supervisor: Rashid Mijumbi

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Gauri Misra		
Student ID:	X20259611		
Programme:	MSc in Cloud Computing	**Year:**	2023
Module:	MSc Research Project		
Lecturer:	Rashid Mijumbi		
Submission Due Date:	14/12/2023		
Project Title:	Anomaly Detection in Cloud System using Novel Aspect of SMOTE Sampling and Machine Learning Classifiers		
Word Count:	457		
Page Count:	7		

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Gauri Misra

Date: 14h December 2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Anomaly Detection in Cloud System using Novel Aspect of SMOTE Sampling and Machine Learning Classifiers

Gauri Misra
x20259611

1 Introduction

The objective of this paper is to provide a detailed explanation of the coding procedure utilized in the project. This document provides an outline of the necessary hardware and software setups required for future replication of the research. This section provides an overview of the steps necessary to execute the script, along with the design and implementation procedures needed to ensure the production of efficient and functional executable code.

2 Hardware Requirments

Hardware Overview:

Model Name:	MacBook Air
Model Identifier:	MacBookAir10,1
Model Number:	MGN63LL/A
Chip:	Apple M1
Total Number of Cores:	8 (4 performance and 4 efficiency)
Memory:	8 GB
System Firmware Version:	10151.41.12
OS Loader Version:	10151.41.12
Serial Number (system):	FVFHRQLPQ6L4
Hardware UUID:	1DF93318-ED94-5799-A530-113001045C9D
Provisioning UDID:	00008103-00060DD614E0801E
Activation Lock Status:	Enabled

3 Software Requirement

For creating the script following tools and softwares were required

Programming Language	Python3.6
Tools	Jupyter Notebook

4 Dataset Collection

For this the dataset has been taken from the following link:

https://www.unb.ca/cic/datasets/nsl.html

5 Importing libraries

```
In [1]: import pandas as pd
        import numpy as np
        from sklearn.preprocessing import OneHotEncoder, LabelEncoder, StandardScaler
        from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score
        from sklearn.ensemble import RandomForestClassifier
        from sklearn.metrics import classification_report, roc_curve, auc
        from sklearn.metrics import confusion_matrix
        from scipy.stats import f_oneway
        from imblearn.over_sampling import SMOTE
        import matplotlib.pyplot as plt
```

Libraries

6 Import Dataset

```
# Load the dataset
data = 'KDDTrain+.txt'
df   = pd.read_csv(data, header=None)
```

7 Data pre-processing

```
In [3]: # Assign generic column names
        column_names = [f'feature_{i}' for i in range(df.shape[1] - 2)] + ['attack_type', 'attack_category']
        df.columns = column_names

        # Check for missing values
        missing_values = df.isnull().sum()
        print(missing_values)
```

```
feature_0         0
feature_1         0
feature_2         0
feature_3         0
feature_4         0
feature_5         0
feature_6         0
feature_7         0
feature_8         0
feature_9         0
feature_10        0
feature_11        0
feature_12        0
feature_13        0
feature_14        0
feature_15        0
feature_16        0
feature_17        0
feature_18        0
feature_19        0
feature_20        0
feature_21        0
feature_22        0
feature_23        0
feature_24        0
feature_25        0
feature_26        0
```

Checking missing values

8 Data transformation

```python
In [4]: # Encode categorical features
encoder = OneHotEncoder(sparse=False)
categorical_columns = df.select_dtypes(include=['object']).columns
encoded_columns = pd.DataFrame(encoder.fit_transform(df[categorical_columns]))
encoded_column_names = encoder.get_feature_names_out(categorical_columns)
encoded_columns.columns = encoded_column_names
df_encoded = df.drop(categorical_columns, axis=1)
df_encoded = pd.concat([df_encoded, encoded_columns], axis=1)

# Normalize numerical features
numerical_columns = df.select_dtypes(include=['int64', 'float64']).columns
scaler = StandardScaler()
df_encoded[numerical_columns] = scaler.fit_transform(df_encoded[numerical_columns])
```

Encoding categorical features

9 Data Visualization

```python
In [5]: # Data Visualization Functions
def plot_histogram(df, column, bins=30):
    plt.figure(figsize=(8, 4))
    plt.hist(df[column], bins=bins, color='skyblue', edgecolor='black')
    plt.title(f'Histogram of {column}')
    plt.xlabel(column)
    plt.ylabel('Frequency')
    plt.show()

def plot_bar_chart(df, column):
    plt.figure(figsize=(8, 4))
    df[column].value_counts().plot(kind='bar', color='skyblue', edgecolor='black')
    plt.title(f'Bar Chart of {column}')
    plt.xlabel(column)
    plt.ylabel('Frequency')
    plt.show()

def plot_box_plot(df, column):
    plt.figure(figsize=(8, 4))
    plt.boxplot(df[column], patch_artist=True, boxprops=dict(facecolor='skyblue'))
    plt.title(f'Box Plot of {column}')
    plt.ylabel(column)
    plt.show()

def plot_scatter_plot(df, column1, column2):
    plt.figure(figsize=(8, 6))
    plt.scatter(df[column1], df[column2], color='skyblue')
    plt.title(f'Scatter Plot of {column1} vs {column2}')
    plt.xlabel(column1)
    plt.ylabel(column2)
    plt.show()

def plot_correlation_heatmap(df):
    corr = df.corr()
    plt.figure(figsize=(12, 10))
```

Data Visualization

10 Data Splitting

```python
In [6]: # ANOVA test function for feature significance
def anova_test(df, numerical_feature, target_feature):
    unique_classes = df[target_feature].unique()
    samples = [df[df[target_feature] == cls][numerical_feature] for cls in unique_classes]
    f_stat, p_value = f_oneway(*samples)
    return f_stat, p_value

# Performing ANOVA test on a feature
f_stat, p_value = anova_test(df, 'feature_4', 'attack_category')

# Encoding target variable for class balancing
le = LabelEncoder()
df_encoded['attack_category_encoded'] = le.fit_transform(df['attack_category'])
X = df_encoded.drop(['attack_category_encoded'], axis=1)
y = df_encoded['attack_category_encoded']

# Split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Apply SMOTE for class balancing
smote = SMOTE(random_state=42)
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)
```

11 Model building and Evaluation

DNN:

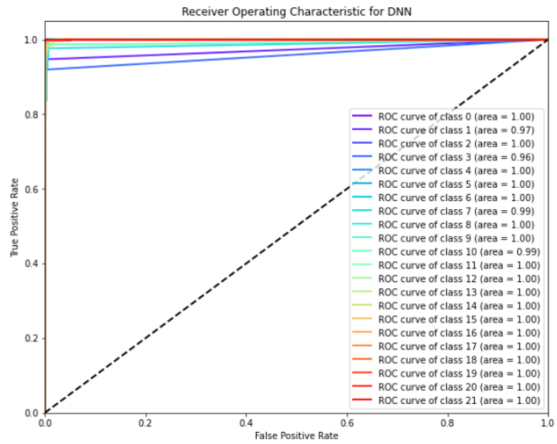
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, SimpleRNN
from tensorflow.keras.utils import to_categorical

# Convert labels to categorical (one-hot encoding)
y_train_smote_cat = to_categorical(y_train_smote)
y_test_cat = to_categorical(y_test)

# Determine the number of unique classes
n_classes = y_train_smote.nunique()

# Create a DNN model
model = Sequential()
model.add(Dense(128, input_shape=(X_train_smote.shape[1],), activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(n_classes, activation='softmax')) # n_classes is the number of unique classes

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Fit the model
model.fit(X_train_smote, y_train_smote_cat, epochs=10, batch_size=32, verbose=1)

# Predict on test data
y_pred_prob = model.predict(X_test)
y_pred_dnn = np.argmax(y_pred_prob, axis=1)

# Classification report
print("DNN Classification Report:")
print(classification_report(y_test, y_pred_dnn))

# Compute ROC curve and ROC area for each class
fpr_dnn, tpr_dnn, roc_auc_dnn = dict(), dict(), dict()

for i in range(n_classes):
    fpr_dnn[i], tpr_dnn[i], _ = roc_curve(y_test_cat[:, i], y_pred_prob[:, i])
    roc_auc_dnn[i] = auc(fpr_dnn[i], tpr_dnn[i])
```


```
print("DNN Classification Report:")
print(classification_report(y_test, y_pred_dnn))

# Compute ROC curve and ROC area for each class
fpr_dnn, tpr_dnn, roc_auc_dnn = dict(), dict(), dict()

for i in range(n_classes):
    fpr_dnn[i], tpr_dnn[i], _ = roc_curve(y_test_cat[:, i], y_pred_prob[:, i])
    roc_auc_dnn[i] = auc(fpr_dnn[i], tpr_dnn[i])
```


```
# Plotting the ROC curve
plt.figure(figsize=(10, 8))
colors = iter(plt.cm.rainbow(np.linspace(0, 1, n_classes)))
for i in range(n_classes):
    plt.plot(fpr_dnn[i], tpr_dnn[i], color=next(colors), lw=2,
             label='ROC curve of class {0} (area = {1:0.2f})'.format(i, roc_auc_dnn[i]))

plt.plot([0, 1], [0, 1], 'k--', lw=2)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic for DNN')
plt.legend(loc="lower right")
plt.show()
```

Applying DNN Model

```
          17       0.98      0.99      0.99       892
          18       1.00      1.00      1.00      6194
          19       1.00      1.00      1.00      3186
          20       1.00      1.00      1.00      5676
          21       1.00      1.00      1.00     18772

    accuracy                           0.99     37792
   macro avg       0.85      0.85      0.85     37792
weighted avg       0.99      0.99      0.99     37792
```



DNN Results

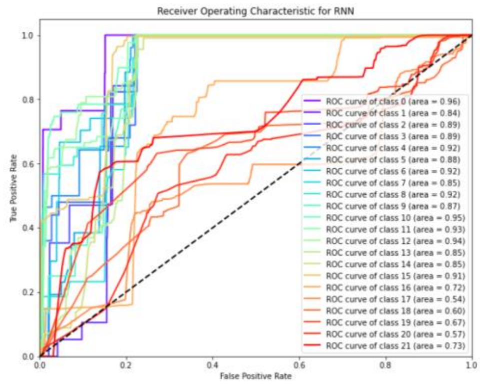
RNN:

In [8]:
```python
# Create RNN model
model = Sequential()
model.add(SimpleRNN(50, return_sequences=True, input_shape=(X_train_smote.shape[1], 1)))
model.add(Dropout(0.2))
model.add(SimpleRNN(50))
model.add(Dropout(0.2))
model.add(Dense(n_classes, activation='softmax'))

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Fit the model
model.fit(X_train_smote, y_train_smote_cat, epochs=10, batch_size=32, verbose=1)

# Predict on test data
y_pred_prob = model.predict(X_test)
y_pred_rnn = np.argmax(y_pred_prob, axis=1)

# Classification report
print("RNN Classification Report:")
print(classification_report(y_test, y_pred_rnn))

# Compute ROC curve and ROC area for each class
fpr_rnn, tpr_rnn, roc_auc_rnn = dict(), dict(), dict()

for i in range(n_classes):
    fpr_rnn[i], tpr_rnn[i], _ = roc_curve(y_test_cat[:, i], y_pred_prob[:, i])
    roc_auc_rnn[i] = auc(fpr_rnn[i], tpr_rnn[i])

# Plotting the ROC curve
plt.figure(figsize=(10, 8))
colors = iter(plt.cm.rainbow(np.linspace(0, 1, n_classes)))
for i in range(n_classes):
    plt.plot(fpr_rnn[i], tpr_rnn[i], color=next(colors), lw=2,
             label='ROC curve of class {0} (area = {1:0.2f})'.format(i, roc_auc_rnn[i]))

plt.plot([0, 1], [0, 1], 'k--', lw=2)
plt.xlim([0.0, 1.0])
```

```python
for i in range(n_classes):
    fpr_rnn[i], tpr_rnn[i], _ = roc_curve(y_test_cat[:, i], y_pred_prob[:, i])
    roc_auc_rnn[i] = auc(fpr_rnn[i], tpr_rnn[i])

# Plotting the ROC curve
plt.figure(figsize=(10, 8))
colors = iter(plt.cm.rainbow(np.linspace(0, 1, n_classes)))
for i in range(n_classes):
    plt.plot(fpr_rnn[i], tpr_rnn[i], color=next(colors), lw=2,
             label='ROC curve of class {0} (area = {1:0.2f})'.format(i, roc_auc_rnn[i]))

plt.plot([0, 1], [0, 1], 'k--', lw=2)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic for RNN')
plt.legend(loc="lower right")
plt.show()
```

RNN Model

18	0.18	0.29	0.22	6194
19	0.18	0.59	0.27	3186
20	0.00	0.00	0.00	5676
21	0.00	0.00	0.00	18772
accuracy			0.14	37792
macro avg	0.03	0.14	0.05	37792
weighted avg	0.05	0.14	0.07	37792

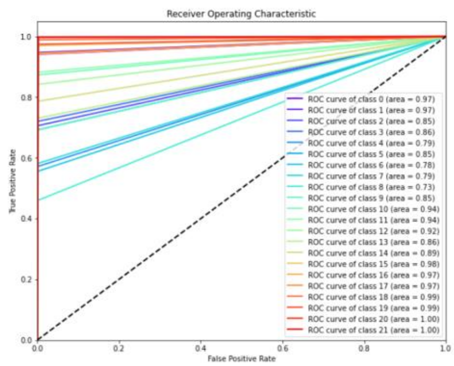


RNN model results

Random forest Classifier

In [9]:
```
# Train RandomForestClassifier
rf = RandomForestClassifier(random_state=42)
rf.fit(X_train_smote, y_train_smote)

# Predict on test data
y_pred = rf.predict(X_test)

# Classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
n_classes = len(np.unique(y_train_smote))

for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test == i, y_pred == i)
    roc_auc[i] = auc(fpr[i], tpr[i])

# Plotting the ROC curve
plt.figure(figsize=(10, 8))
colors = iter(plt.cm.rainbow(np.linspace(0, 1, n_classes)))
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], color=next(colors), lw=2, label='ROC curve of class {0} (area = {1:0.2f})'.format(i, roc_auc[i]))

plt.plot([0, 1], [0, 1], 'k--', lw=2)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()
```

18	0.99	0.99	0.99	6194
19	0.98	0.97	0.98	3186
20	0.98	1.00	0.99	5676
21	1.00	1.00	1.00	18772
accuracy			0.98	37792
macro avg	0.82	0.81	0.81	37792
weighted avg	0.98	0.98	0.98	37792



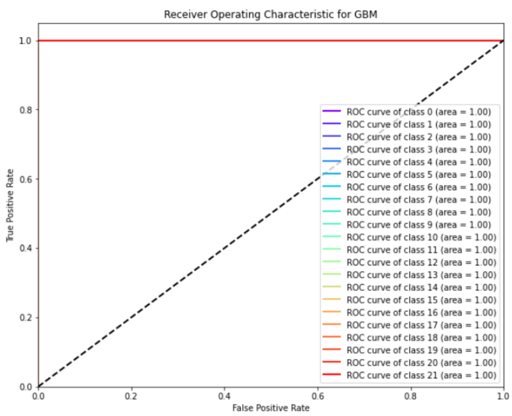
RF results

Gradient Boosting Classifier

```python
In [10]: from sklearn.ensemble import GradientBoostingClassifier

         # Train GradientBoostingClassifier
         gbm = GradientBoostingClassifier(random_state=42)
         gbm.fit(X_train_smote, y_train_smote)

         # Predict on test data
         y_pred_gbm = gbm.predict(X_test)

         # Classification report
         print("GBM Classification Report:")
         print(classification_report(y_test, y_pred_gbm))

         # Compute ROC curve and ROC area for each class
         fpr_gbm, tpr_gbm, roc_auc_gbm = dict(), dict(), dict()
         n_classes = len(np.unique(y_train_smote))

         for i in range(n_classes):
             fpr_gbm[i], tpr_gbm[i], _ = roc_curve(y_test == i, y_pred_gbm == i)
             roc_auc_gbm[i] = auc(fpr_gbm[i], tpr_gbm[i])

         # Plotting the ROC curve
         plt.figure(figsize=(10, 8))
         colors = iter(plt.cm.rainbow(np.linspace(0, 1, n_classes)))
         for i in range(n_classes):
             plt.plot(fpr_gbm[i], tpr_gbm[i], color=next(colors), lw=2,
                      label='ROC curve of class {0} (area = {1:0.2f})'.format(i, roc_auc_gbm[i]))

         plt.plot([0, 1], [0, 1], 'k--', lw=2)
         plt.xlim([0.0, 1.0])
         plt.ylim([0.0, 1.05])
         plt.xlabel('False Positive Rate')
         plt.ylabel('True Positive Rate')
         plt.title('Receiver Operating Characteristic for GBM')
         plt.legend(loc="lower right")
         plt.show()
```

```
GBM Classification Report:
```

19	1.00	1.00	1.00	5186
20	1.00	1.00	1.00	5676
21	1.00	1.00	1.00	18772
accuracy			1.00	37792
macro avg	1.00	1.00	1.00	37792
weighted avg	1.00	1.00	1.00	37792



GBM Results