Configuration Manual

MSc Research Project
MSc Data Analytics 2023-2024

Arya Eldho
Student ID: 22248056

School of Computing
National College of Ireland

Supervisor: Jorge Basilio

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: ……. ……ARYA ELDHO……

Student ID: ……22248056……..……

Programme: ……MSC DATA ANALYTICS…… **Year:** ……2023-2024.

Module: ……MSC RESEARCH PROJECT……..……

Lecturer: ……JORGE BASILIO……..……

Submission Due Date: ……12 AUGUST 2024……..……

Project Title: …… ADVANCING NETWORK INTRUSION DETECTION SYSTEMS THROUGH MACHINE LEARNING ALGORITHMS ……..……

page Count: ……12……..

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: ……ARYA ELDHO……

Date: ……..12 AUGUST 2024……

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):	

Configuration Manual

Arya Eldho
Student ID: 22248056

1 Introduction

The configuration manual provides the complete details of the system configuration, tools and algorithms used for the advancement of network intrusion detection systems using machine learning algorithms. In the research study a machine learning based IDS is built. The machine learning models :SVM, Random Forest(RF), KNN and a hybrid deep learning model and the Convolutional Neural Network-Gated Recurrent Unit-Bidirectional Long Short Term Memory(CNN-GRU-BiLSTM) models are used in the study. The dimensionality of the features in the dataset is reduced using both Recursive Feature Elimination (RFE) and Principal Component Analysis(PCA). To validate the performance of the models, confusion matrix, evaluation matrix and ROC curves were used. The process analyzed during the development phase and all the research findings are documented in the following sections.

2. System Specification

The hardware and software requirements are mentioned in the following section

2.1 Hardware Requirements

Operating system	Windows 11
RAM	16 GB
Hard Disk	250 GB

2.2 Software Requirements

The research work was implemented using machine learning models, and hence R studio is used for the implementation.

3. Data Source

The dataset used for the study is NSL-KDD dataset taken from the Kaggle. The data source was very useful in evaluating network intrusions. The dataset is downloaded from the given source link : https://www.kaggle.com/datasets/hassan06/nslkdd

4. Data Load And Analysis

First step was to import all the necessary libraries as shown in fig 1.

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE
from collections import Counter
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score,precision_score
import joblib
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib import pyplot
import operator
import tensorflow as tf
import pickle
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from mlxtend.plotting import plot_confusion_matrix
from tensorflow.keras.callbacks import ModelCheckpoint
from replace import *
from sklearn.preprocessing import LabelEncoder
```

Fig 1: Importing libraries

Next process is to load the train and test data from the CSV and to assign the column names as ‘feature’

```
print("Train")
feature=["duration","protocol_type","service","flag","src_bytes","dst_bytes","land","wrong_fragment","urge
         "num_failed_logins","logged_in","num_compromised","root_shell","su_attempted","num_root","num_fi
         "num_access_files","num_outbound_cmds","is_host_login","is_guest_login","count","srv_count","ser
         "rerror_rate","srv_rerror_rate","same_srv_rate","diff_srv_rate","srv_diff_host_rate","dst_host_c
         "dst_host_same_srv_rate","dst_host_diff_srv_rate","dst_host_same_src_port_rate","dst_host_srv_di
         "dst_host_srv_serror_rate","dst_host_rerror_rate","dst_host_srv_rerror_rate","attack","level"]


train='Dataset/KDDTrain+.txt'
test='Dataset/KDDTest+.txt'
train_data=pd.read_csv(train,names=feature)
test_data=pd.read_csv(test,names=feature)
```

Fig 2: feature name definition and data loading

5. Data Processing

This section explains the preprocessing of the dataset. Here the dataset summary is driven, dropping the columns, and label modification.

```
train='Dataset/KDDTrain+.txt'
test='Dataset/KDDTest+.txt'
train_data=pd.read_csv(train,names=feature)
test_data=pd.read_csv(test,names=feature)
print("################################")
print(train_data)
print("################################")
train_data.drop(['level'],axis=1,inplace=True)
print(train_data.shape)
print("################################")
print(train_data.info())
print("################################")
print(train_data['attack'].value_counts())
print("################################")
```

Fig 3: Basic data exploration

```
def change_label(df):
  df.attack.replace(['apache2','back','land','neptune','mailbomb','pod','processtable','smurf','teardrop',
  df.attack.replace(['ftp_write','guess_passwd','httptunnel','imap','multihop','named','phf','sendmail','s
  df.attack.replace(['ipsweep','mscan','nmap','portsweep','saint','satan'],'Probe',inplace=True)
  df.attack.replace(['buffer_overflow','loadmodule','perl','ps','rootkit','sqlattack','xterm'],'U2R',inpla

change_label(train_data)

train_data.to_csv('Dataset/train.csv',index=False)
print('##############################')
print(train_data['attack'].value_counts())
print('##############################')
################################################################################################
print("Test")
print("################################")
print(test_data)
print("################################")
test_data.drop(['level'],axis=1,inplace=True)
print(test_data.shape)
print("################################")
print(test_data.info())
```

Fig 4: label modification

6. Model Building And Evalutaion

For the implementation process, the study has used Support Vector Machine (SVM), Random Forest, K-Nearest Neighbors, Isolation forest and Convolutional Neural Network-Gated Recurrent Unit-Long Short-Term Memory (CNN-GRU-LSTM) to train in three different scenarios : without any feature reduction method, with features reduced by RFE and PCA. The technique SMOTE is used to generate samples to balance the distribution.

6.1 Model training without feature dimensionality reduction methods

Traditional models such as KNN, random forest and SVM are trained without using feature directionality reduction methods.

```
#Data balancing using SMOTE
counter = Counter(y_train)
print("Before Balancing :", counter)

smt = SMOTE(k_neighbors=1)

x_train_sm, y_train_sm = smt.fit_resample(x_train, y_train)

counter = Counter(y_train_sm)
print("After Balancing : ", counter)
##########################################################################################

scaler = StandardScaler()
x_train_sm = scaler.fit_transform(x_train_sm)
x_test = scaler.transform(x_test)
print(x_train_sm.shape, y_train_sm.shape, x_test.shape, y_test.shape)
pickle.dump(scaler,open('NSL_KDD_WithoutFe/scaler_nsl_traditional.pkl','wb'))
```

Fig 5 : data balancing using SMOTE

Training the random forest model without any feature dimensionality reduction method

```
# intialize randomforest classifier
clas1 = RandomForestClassifier(n_estimators = 100)
#Training
clas1.fit(x_train_sm, y_train_sm)
y_pred1 = clas1.predict(x_test)
```

Fig 6: Model 1 : Random Forest Classifier

Evaluation matrix generated with random forest classifier

```
#cm
conf_matrix1 = confusion_matrix(y_true=y_test, y_pred=y_pred1)
fig, ax = plot_confusion_matrix(conf_mat=conf_matrix1, figsize=(6, 6), cmap=plt.cm.OrRd)
plt.xlabel('Predictions', fontsize=18)
plt.ylabel('Actuals', fontsize=18)
plt.title('Confusion Matrix', fontsize=18)
plt.savefig('NSL_KDD_WithoutFe/c_matrix_rf.png')
plt.show()
acc1 = accuracy_score(y_test, y_pred1)
print(f"RF Accuracy :{round(acc1,3)*100}%")


#accuracy
# precision
pre1 = precision_score(y_test, y_pred1, average='weighted')
print(f"RF Precision  :{round(pre1,3)*100}%")

# model saving
fn1 = "NSL_KDD_WithoutFe/RF_Model.sav"
joblib.dump(clas1, fn1)
```

Fig 7: Random forest evaluation

Training and prediction of KNN classifier without feature dimensionality reduction method

```
# Initialize KNN classifier
knn = KNeighborsClassifier(n_neighbors=5)

# Training
knn.fit(x_train_sm, y_train_sm)

# Prediction
y_pred_knn = knn.predict(x_test)
```

Fig 8 : Model 2 : KNN

```
conf_matrix1 = confusion_matrix(y_true=y_test, y_pred=y_pred_knn)
fig, ax = plot_confusion_matrix(conf_mat=conf_matrix1, figsize=(6, 6), cmap=plt.cm.OrRd)
plt.xlabel('Predictions', fontsize=18)
plt.ylabel('Actuals', fontsize=18)
plt.title('Confusion Matrix', fontsize=18)
plt.savefig('NSL_KDD_WithoutFe/c_matrix_knn.png')
plt.show()

# Accuracy
acc_knn = accuracy_score(y_test, y_pred_knn)
print(f"KNN Accuracy: {round(acc_knn, 3)*100}%")

# Precision
pre_knn = precision_score(y_test, y_pred1, average='weighted')
print(f"KNN Precision: {round(pre_knn, 3)*100}%")

# Model saving
fn_knn = "NSL_KDD_WithoutFe/KNN_Model.sav"
joblib.dump(knn, fn_knn)
```

Fig 9 : evaluation of KNN

Training of SVM classifier without feature dimensionality reduction method

```
from sklearn.svm import SVC

# Initialize SVM classifier
svm = SVC(kernel='linear', C=1.0)

# Training
svm.fit(x_train_sm, y_train_sm)

# Prediction
y_pred_svm = svm.predict(x_test)
```

Fig 10 : model 3: SVM model

```
conf_matrix1 = confusion_matrix(y_true=y_test, y_pred=y_pred_svm)
fig, ax = plot_confusion_matrix(conf_mat=conf_matrix1, figsize=(6, 6), cmap=plt.cm.OrRd)
plt.xlabel('Predictions', fontsize=18)
plt.ylabel('Actuals', fontsize=18)
plt.title('Confusion Matrix', fontsize=18)
plt.savefig('NSL_KDD_WithoutFe/c_matrix_svm.png')
plt.show()
# Accuracy
acc_svm = accuracy_score(y_test, y_pred_svm)
print(f"SVM Accuracy: {round(acc_svm, 3)*100}%")

# Precision
pre_svm = precision_score(y_test, y_pred1, average='weighted')
print(f"SVM Precision: {round(pre_svm, 3)*100}%")

# Model saving
fn_svm = "NSL_KDD_WithoutFe/SVM_Model.sav"
joblib.dump(svm, fn_svm)
```

Fig 11 : svm evaluation

Training of neural model without feature dimensionality reduction method

```
# Define CNN-GRU model
model = Sequential([
    Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=(x_train_cnn.shape[1], 1)),
    MaxPooling1D(pool_size=2),
    Flatten(),
    Dense(50, activation='relu'),
    tf.keras.layers.Reshape((1, 50)),
    GRU(50, return_sequences=True),
    Bidirectional(LSTM(50)),
    Dense(5, activation='softmax')
])
# Compile the model
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Define callbacks
checkpoint = ModelCheckpoint('NSL_KDD_WithoutFe/model_cnn_gru.h5', monitor='val_accuracy', verbose=1, sav
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

# Train the model
history = model.fit(x_train_cnn, y_train_sm, epochs=20, batch_size=64, validation_split=0.2, callbacks=[c
```

Fig 12 : training CNN-GRU model

```
# Plot training and validation accuracy
plt.plot(history.history['accuracy'], label='train_accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0, 1])
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')
plt.show()

# Plot training and validation loss
plt.plot(history.history['loss'], label='train_loss')
plt.plot(history.history['val_loss'], label='val_loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```

Fig 13 : CNN-GRU model evaluation

6.2 Model training with PCA

Model training using principal component analysis (PCA) to handle the multicollinearity of the data retaining the data accuracy.

```
# intialize randomforest classifier
clas1 = RandomForestClassifier(n_estimators = 100)
#Training
clas1.fit(x_train_sm_pca, y_train_sm)
y_pred1 = clas1.predict(x_test_pca)

#cm
conf_matrix1 = confusion_matrix(y_true=y_test, y_pred=y_pred1)
fig, ax = plot_confusion_matrix(conf_mat=conf_matrix1, figsize=(6, 6), cmap=plt.cm.OrRd)
plt.xlabel('Predictions', fontsize=18)
plt.ylabel('Actuals', fontsize=18)
plt.title('Confusion Matrix', fontsize=18)
plt.savefig('NSL_KDD_PCA/c_matrix_rf.png')
plt.show()
acc1 = accuracy_score(y_test, y_pred1)
print(f"RF Accuracy :{round(acc1,3)*100}%")
#accuracy
# precision
pre1 = precision_score(y_test, y_pred1, average='weighted')
print(f"RF Precision  :{round(pre1,3)*100}%")

# model saving
fn1 = "NSL_KDD_PCA/RF_Model.sav"
joblib.dump(clas1, fn1)
```

Fig 14 : Random forest model training and evaluation with PCA

```
# Initialize KNN classifier
knn = KNeighborsClassifier(n_neighbors=5)
# Training
knn.fit(x_train_sm_pca, y_train_sm)
# Prediction
y_pred_knn = knn.predict(x_test_pca)

conf_matrix1 = confusion_matrix(y_true=y_test, y_pred=y_pred_knn)
fig, ax = plot_confusion_matrix(conf_mat=conf_matrix1, figsize=(6, 6), cmap=plt.cm.OrRd)
plt.xlabel('Predictions', fontsize=18)
plt.ylabel('Actuals', fontsize=18)
plt.title('Confusion Matrix', fontsize=18)
plt.savefig('NSL_KDD_PCA/c_matrix_knn.png')
plt.show()
# Accuracy
acc_knn = accuracy_score(y_test, y_pred_knn)
print(f"KNN Accuracy: {round(acc_knn, 3)*100}%")
# Precision
pre_knn = precision_score(y_test, y_pred1, average='weighted')
print(f"KNN Precision: {round(pre_knn, 3)*100}%")
# Model saving
fn_knn = "NSL_KDD_PCA/KNN_Model.sav"
joblib.dump(knn, fn_knn)
```

Fig 15 : KNN model training with PCA and evaluation

SVM model training with PCA

```
# Initialize SVM classifier
svm = SVC(kernel='linear', C=1.0)
# Training
svm.fit(x_train_sm_pca, y_train_sm)
# Prediction
y_pred_svm = svm.predict(x_test_pca)
conf_matrix1 = confusion_matrix(y_true=y_test, y_pred=y_pred_svm)
fig, ax = plot_confusion_matrix(conf_mat=conf_matrix1, figsize=(6, 6), cmap=plt.cm.OrRd)
plt.xlabel('Predictions', fontsize=18)
plt.ylabel('Actuals', fontsize=18)
plt.title('Confusion Matrix', fontsize=18)
plt.savefig('NSL_KDD_PCA/c_matrix_svm.png')
plt.show()
# Accuracy
acc_svm = accuracy_score(y_test, y_pred_svm)
print(f"SVM Accuracy: {round(acc_svm, 3)*100}%")
# Precision
pre_svm = precision_score(y_test, y_pred1, average='weighted')
print(f"SVM Precision: {round(pre_svm, 3)*100}%")
# Model saving
fn_svm = "NSL_KDD_PCA/SVM_Model.sav"
joblib.dump(svm, fn_svm)
```

Fig 16 : SVM training and evaluation

CNN-GRU model training and evaluation with PCA

```
# Define CNN-GRU model
model = Sequential([
    Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=(x_train_cnn.shape[1], 1)),
    MaxPooling1D(pool_size=2),
    Flatten(),
    Dense(50, activation='relu'),
    tf.keras.layers.Reshape((1, 50)),
    GRU(50, return_sequences=True),
    Bidirectional(LSTM(50)),
    Dense(5, activation='softmax')
])
# Compile the model
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
# Define callbacks
checkpoint = ModelCheckpoint('NSL_KDD_PCA/model_cnn_gru.h5', monitor='val_accuracy', verbose=1, save_best
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
# Train the model
history = model.fit(x_train_cnn, y_train_sm, epochs=20, batch_size=64, validation_split=0.2, callbacks=[c
```

Fig 17 : CNN -GRU model training

```
# Plot training and validation accuracy
plt.plot(history.history['accuracy'], label='train_accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0, 1])
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')
plt.show()
# Plot training and validation loss
plt.plot(history.history['loss'], label='train_loss')
plt.plot(history.history['val_loss'], label='val_loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
##########################################################################################################
# Predictions on test data
y_pred = model.predict(x_test_cnn)
y_pred_classes = np.argmax(y_pred, axis=1)
# Confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred_classes)
```

Fig 18 : CNN-GRU model evaluation

6.3 Models with RFE

The models are trainied with Recurssive feauture elimation (RFE) to improve the model interpretability.

```
# intialize randomforest classifier
clas1 = RandomForestClassifier(n_estimators = 100)
#Training
clas1.fit(x_train_sm, y_train_sm)
y_pred1 = clas1.predict(x_test)
#cm
conf_matrix1 = confusion_matrix(y_true=y_test, y_pred=y_pred1)
fig, ax = plot_confusion_matrix(conf_mat=conf_matrix1, figsize=(6, 6), cmap=plt.cm.OrRd)
plt.xlabel('Predictions', fontsize=18)
plt.ylabel('Actuals', fontsize=18)
plt.title('Confusion Matrix', fontsize=18)
plt.savefig('NSL_KDD_RFE/c_matrix_rf.png')
plt.show()
acc1 = accuracy_score(y_test, y_pred1)
print(f"RF Accuracy :{round(acc1,3)*100}%")
#accuracy
# precision
pre1 = precision_score(y_test, y_pred1, average='weighted')
print(f"RF Precision  :{round(pre1,3)*100}%")
# model saving
fn1 = "NSL_KDD_RFE/RF_Model.sav"
joblib.dump(clas1, fn1)
```

Fig 19 : random forest model training and evaluation with RFE

```
# Initialize KNN classifier
knn = KNeighborsClassifier(n_neighbors=5)
# Training
knn.fit(x_train_sm, y_train_sm)
# Prediction
y_pred_knn = knn.predict(x_test)
conf_matrix1 = confusion_matrix(y_true=y_test, y_pred=y_pred_knn)
fig, ax = plot_confusion_matrix(conf_mat=conf_matrix1, figsize=(6, 6), cmap=plt.cm.OrRd)
plt.xlabel('Predictions', fontsize=18)
plt.ylabel('Actuals', fontsize=18)
plt.title('Confusion Matrix', fontsize=18)
plt.savefig('NSL_KDD_RFE/c_matrix_knn.png')
plt.show()
# Accuracy
acc_knn = accuracy_score(y_test, y_pred_knn)
print(f"KNN Accuracy: {round(acc_knn, 3)*100}%")
# Precision
pre_knn = precision_score(y_test, y_pred1, average='weighted')
print(f"KNN Precision: {round(pre_knn, 3)*100}%")
# Model saving
fn_knn = "NSL_KDD_RFE/KNN_Model.sav"
joblib.dump(knn, fn_knn)
```

Fig 20: KNN model training and evaluation with RFE

```
# Initialize SVM classifier
svm = SVC(kernel='linear', C=1.0)
# Training
svm.fit(x_train_sm, y_train_sm)
# Prediction
y_pred_svm = svm.predict(x_test)
conf_matrix1 = confusion_matrix(y_true=y_test, y_pred=y_pred_svm)
fig, ax = plot_confusion_matrix(conf_mat=conf_matrix1, figsize=(6, 6), cmap=plt.cm.OrRd)
plt.xlabel('Predictions', fontsize=18)
plt.ylabel('Actuals', fontsize=18)
plt.title('Confusion Matrix', fontsize=18)
plt.savefig('NSL_KDD_RFE/c_matrix_svm.png')
plt.show()
# Accuracy
acc_svm = accuracy_score(y_test, y_pred_svm)
print(f"SVM Accuracy: {round(acc_svm, 3)*100}%")
# Precision
pre_svm = precision_score(y_test, y_pred1, average='weighted')
print(f"SVM Precision: {round(pre_svm, 3)*100}%")
# Model saving
fn_svm = "NSL_KDD_RFE/SVM_Model.sav"
joblib.dump(svm, fn_svm)
```

Fig 21 : SVM model training and evaluation with RFE

```
# Define CNN-GRU model
model = Sequential([
    Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=(x_train_cnn.shape[1], 1)),
    MaxPooling1D(pool_size=2),
    Flatten(),
    Dense(50, activation='relu'),
    tf.keras.layers.Reshape((1, 50)),
    GRU(50, return_sequences=True),
    Bidirectional(LSTM(50)),
    Dense(5, activation='softmax')
])
# Compile the model
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Define callbacks
checkpoint = ModelCheckpoint('NSL_KDD_RFE/model_cnn_gru.h5', monitor='val_accuracy', verbose=1, save_best
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

# Train the model
history = model.fit(x_train_cnn, y_train, epochs=10, batch_size=64, validation_split=0.2, callbacks=[chec
```

Fig 22 : CNN-GRU model training with RFE

```
# Plot training and validation accuracy
plt.plot(history.history['accuracy'], label='train_accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0, 1])
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')
plt.show()
# Plot training and validation loss
plt.plot(history.history['loss'], label='train_loss')
plt.plot(history.history['val_loss'], label='val_loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
##########################################################################################################
# Predictions on test data
y_pred = model.predict(x_test_cnn)
y_pred_classes = np.argmax(y_pred, axis=1)
# Confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred_classes)
```

Fig 23: CNN-GRU model evalutaion

6.4 GUI Interface for prediction

As part of the research study, the desktop application developed here utilizes the CNN-GRU-BiLSTM model, provides a user friendly solution for the network intrusion. The UI is developed using python libraries such as Tkinter. Here the users can input the network traffic data in various formats. The preprocessed data is fed into the model, generate the output window by predicting the intrusion type.

```
# Load the trained label encoder
label_encoder = joblib.load('NSL_KDD_RFE/label_encoder_withoutfe_neu.pkl')

# Load the trained CNN-GRU model
model = load_model('NSL_KDD_RFE/model_cnn_gru.h5')

def button_click():
    text_content = text_area.get("1.0", "end-1c")  # Get content from text area
    print(text_content)
    # Define the input data
    x_final_pred = np.array([0, 18, 491, 0, 0, 2, 25, 0.03, 0.17, 0]).reshape(1, -1)

    # Scale the input features
    x_final_pred_scaled = scaler.transform(x_final_pred)

    # Reshape input data for CNN
    x_final_pred_cnn = x_final_pred_scaled.reshape(x_final_pred_scaled.shape[0], x_final_pred_scaled.shap

    # Make predictions
    y_pred = model.predict(x_final_pred_cnn)
    y_pred_classes = np.argmax(y_pred, axis=1)

    # Inverse transform the predicted labels
    predicted_labels = label_encoder.inverse_transform(y_pred_classes)
    print(predicted_labels)

root = tk.Tk()
root.resizable(False, False)

text_area = tk.Text(root, height=10, width=50)
text_area.pack(padx=10, pady=10)

button = tk.Button(root, text="Predict", command=button_click)
button.pack(pady=5)

root.mainloop()
```

Fig 24: GUI Interface for prediction