National College of Ireland

Configuration Manual for An Artificial Intelligence aided simulation testing framework for network intrusion detection in different operating systems

MSc Research Project
Cyber Security

Sri Lakshmi Pamarthi
Student ID: x23142294

National College of Ireland

Supervisor: Jawad Salahuddin

National College of Ireland

MSc Project Submission Sheet

Student Name: ……Sri Lakshmi Pamarthi……

Student ID: ……x23142294……..……

Programme: ……Cyber Security…… **Year:** ……2023-24..

Module: ……MSc Research Project……..……

Lecturer: …… Jawad Salahuddin ……..……

Submission Due Date: ……12-08-2024……..……

Project Title: ..... An Artificial Intelligence aided simulation testing framework for network intrusion detection in different operating systems ...

Word Count: ……611…… **Page Count:** ……9……..……..……

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: ……Srilakshmi Pamarthi……

Date: ……12-08-2024……

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 for Agile

1 System Requirements

This whole project takes into the account three important steps,

RAM: 8GB DDR2

OS: Windows 11

Processor: i5 13th generation

Technology required: Python, Anaconda prompt, Spyder, Streamlit

2 Code execution

```
#loading and importing data
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split,StratifiedKFold,GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier,VotingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import GridSearchCV,RandomizedSearchCV
from sklearn.svm import SVC
from sklearn import metrics
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import make_scorer, roc_auc_score
import scipy
from scipy import stats
import warnings
warnings.filterwarnings("ignore")
from sklearn.preprocessing import StandardScaler
```

Figure1. This script imports necessary packages and functions for processing data as well as for generating and evaluating the models for machine learning and statistical computations. This includes; numpy, pandas, matplotlib, seaborn, several classes and functions from the sklearn package.

```
# add the column labels
columns = (['duration'
,'protocol_type'
,'service'
,'flag'
,'src_bytes'
,'dst_bytes'
,'land'
,'wrong_fragment'
,'urgent'
,'hot'
,'num_failed_logins'
,'logged_in'
,'num_compromised'
,'root_shell'
,'su_attempted'
,'num_root'
,'num_file_creations'
,'num_shells'
,'num_access_files'
,'num_outbound_cmds'
,'is_host_login'
,'is_guest_login'
,'count'
,'srv_count'
,'serror_rate'
,'srv_serror_rate'
,'rerror_rate'
,'srv_rerror_rate'
,'same_srv_rate'
,'diff_srv_rate'
,'srv_diff_host_rate'
,'dst_host_count'
,'dst_host_srv_count'
,'dst_host_same_srv_rate'
,'dst_host_diff_srv_rate'
,'dst_host_same_src_port_rate'
,'dst_host_srv_diff_host_rate'
,'dst_host_serror_rate'
,'dst_host_srv_serror_rate'
,'dst_host_rerror_rate'
,'dst_host_srv_rerror_rate'
,'attack'
,'level'])


df_train=pd.read_csv('/content/drive/MyDrive/nsl_kdd/KDDTrain+.txt',header=None,names=columns)
df_test=pd.read_csv('/content/drive/MyDrive/nsl_kdd/KDDTest+.txt',header=None,names=columns)
```

Figure2. This script tends to generate a list of the 15 variables; such as ‘duration’, ‘protocol_type’, ‘service’, ‘flag’, ‘src_bytes’, ‘dst_bytes’, ‘land’, ‘wrong_fragment’, and others, which are the feature identifiers for a dataset. It also sets out datasets for training/learning and assessment purposes.

```
print(df_train.duplicated().sum())
print(df_test.duplicated().sum())
```

```
df_train.isnull().sum()
```

```
df_train['attack'].value_counts()
```

```
df_train["binary_attack"]=df_train.attack.map(lambda a: "normal" if a == 'normal' else "abnormal")
df_train.drop('attack',axis=1,inplace=True)

df_test["binary_attack"]=df_test.attack.map(lambda a: "normal" if a == 'normal' else "abnormal")
df_test.drop('attack',axis=1,inplace=True)
```

```
x_train=df_train.drop('binary_attack',axis=1)
y_train=df_train["binary_attack"]

x_test=df_test.drop('binary_attack',axis=1)
y_test=df_test["binary_attack"]
```

Figure3. We split samples into observed or predictor variables/feature variables (also known as the independent variables or explanatory variables) or often denoted by X and response or dependent variables or label data or often denoted by Y.

```
from sklearn.feature_selection import mutual_info_classif
mutual_info = mutual_info_classif(x_train, y_train)
mutual_info = pd.Series(mutual_info)
mutual_info.index = x_train.columns
mutual_info.sort_values(ascending=False)
```

Figure4. The present script incorporates `LabelEncoder` from `sklearn. preprocessing to the necessary data format to convert categories into numbers. It traverses over the mentioned column names namely, 'protocol_type,' 'service,' 'flag', and 'binary_attack,' for both the training set and the test set and performs the label encoding for each of the columns present in the data frame 'df_train' and 'df_test'.

```
from sklearn.feature_selection import SelectKBest
sel_five_cols = SelectKBest(mutual_info_classif, k=20)
sel_five_cols.fit(x_train, y_train)
x_train.columns[sel_five_cols.get_support()]
```

Figure5. SelectKBest function is employed here with the scoring function of `mutual_info_classif` to select the 20 most essential features from the training data set. It applies this selection on `x_train`

and `y_train` and gets the names of the columns of the selected features from `x_train`.

```
col=['service', 'flag', 'src_bytes', 'dst_bytes', 'logged_in',
       'same_srv_rate', 'diff_srv_rate', 'dst_host_srv_count',
       'dst_host_same_srv_rate', 'dst_host_diff_srv_rate']
x_train=x_train[col]
x_test=x_test[col]
```

Figure6. This script prepares the two predictors, `x_train` and `x_test`, by reducing the latter to the columns stated in `col`. This is done to guarantee that not only the training dataset, but also the testing dataset only include the features from the `col` list which are 'service,' 'flag,' 'src_bytes,' 'dst_bytes,' 'logged_in,' 'same_srv_rate,' 'diff_srv_rate,' 'dst_host_srv_count,' 'dst_host_same_srv_rate,' and 'dst_host_diff_srv_rate. '

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
x_train= scaler.fit_transform(x_train)
x_test= scaler.fit_transform(x_test)
```

Figure7. This script uses the `MinMaxScaler` preprocessing class from `sklearn. The first step of transformations is to perform the 'preprocessing' on the features in `x_train` and `x_test` to scale it between 0 and 1. It fits the scaler on `x_train` and as the next step applies transformation on both the datasets.

```python
models = {}

# Logistic Regression
from sklearn.linear_model import LogisticRegression
models['Logistic Regression'] = LogisticRegression()

# Support Vector Machines
from sklearn.svm import LinearSVC
models['Support Vector Machines linear'] = LinearSVC()
models['Support Vector Machines plonomial'] = SVC(kernel='poly')
models['Support Vector Machines RBf'] =  SVC(C=100.0)

# Decision Trees
from sklearn.tree import DecisionTreeClassifier
models['Decision Trees'] = DecisionTreeClassifier(max_depth=3)

# Random Forest
from sklearn.ensemble import RandomForestClassifier
models['Random Forest'] = RandomForestClassifier()

# Naive Bayes
from sklearn.naive_bayes import GaussianNB
models['Naive Bayes'] = GaussianNB()

# K-Nearest Neighbors
from sklearn.neighbors import KNeighborsClassifier
models['K-Nearest Neighbor'] = KNeighborsClassifier(n_neighbors=20)
```

Figure8.. This script defines several machine learning models and saves them into the `models` list. It contains Logistic Regression, SVM – Linear, Polynomial, Radial basis function, Decision Trees, Random Forest, Naive Bayes, K-Nearest Neighbor and these algorithms have their own set of parameters.

```python
from sklearn.metrics import accuracy_score, precision_score, recall_score

accuracy, precision, recall = {}, {}, {}

for key in models.keys():

    # Fit the classifier
    models[key].fit(x_train, y_train)

    # Make predictions
    predictions = models[key].predict(x_test)

    # Calculate metrics
    accuracy[key] = accuracy_score(predictions, y_test)
    precision[key] = precision_score(predictions, y_test)
    recall[key] = recall_score(predictions, y_test)
```

FigureG. This script takes `x_train`, and `y_train` to train each individual model in `models`, and then tests the model on `x_test`, and finally calculates accuracy, precision and recall. These metrics are held in their often respective metrics dictionaries.

```python
df_model = pd.DataFrame(index=models.keys(), columns=['Accuracy', 'Precision', 'Recall'])
df_model['Accuracy'] = accuracy.values()
df_model['Precision'] = precision.values()
df_model['Recall'] = recall.values()

df_model
```

Figure10. This script defines a DataFrame named `df_model`, in which every row contains a model while the column names of the DataFrame are 'Accuracy', 'Precision', and 'Recall'. These it fills with data from the `accuracy`, `precision`, and `recall` dictionaries that were defined earlier in this code.

```python
ax = df_model.plot.barh()
ax.legend(
    ncol=len(models.keys()),
    bbox_to_anchor=(0, 1),
    loc='lower left',
    prop={'size': 14}
)
plt.tight_layout()
```

Figure11. This script barplots from `df_model` and adds a legend of the length for the number of models, placing it in the lower left corner, changing the font size of the legend and advises a tight layout.

```
C:\Users\srila>cd C:\THESIS\MAIN_PYTHON

C:\THESIS\MAIN_PYTHON>pip install streamlit
```

Figure12: This command is used to install the streamlit

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
```

```
model_filepath = os.path.join(os.path.dirname(__file__), "classifier.pkl")
data_path = os.path.join(os.path.dirname(__file__),"test.csv")
train = os.path.join(os.path.dirname(__file__),"KDDTrain+.txt")
test = os.path.join(os.path.dirname(__file__),"KDDTest+.txt")

def trained_model():
    # Load dataset (using Iris dataset as an example)
    with open(train, 'r') as file:
        # Read the entire content of the file
        data = file.read()

    X_train, X_val, y_train, y_val = train_test_split(data.data, data.target, test_size=0.3, random_state=42)

    # Train models
    log_reg = LogisticRegression(max_iter=1000)
    log_reg.fit(X_train, y_train)

    random_forest = RandomForestClassifier()
    random_forest.fit(X_train, y_train)

    xgboost_model = xgb.XGBClassifier(use_label_encoder=False, eval_metric='logloss')
    xgboost_model.fit(X_train, y_train)

    lgb_model = lgb.LGBMClassifier()
    lgb_model.fit(X_train, y_train)
```

Figure13

```
# Train models
log_reg = LogisticRegression(max_iter=1000)
log_reg.fit(X_train, y_train)

random_forest = RandomForestClassifier()
random_forest.fit(X_train, y_train)

xgboost_model = xgb.XGBClassifier(use_label_encoder=False, eval_metric='logloss')
xgboost_model.fit(X_train, y_train)

lgb_model = lgb.LGBMClassifier()
lgb_model.fit(X_train, y_train)

# Predictions
log_reg_pred = log_reg.predict(X_val)
random_forest_pred = random_forest.predict(X_val)
xgboost_pred = xgboost_model.predict(X_val)
lgb_pred = lgb_model.predict(X_val)

# Evaluation
log_reg_acc = accuracy_score(y_val, log_reg_pred)
random_forest_acc = accuracy_score(y_val, random_forest_pred)
xgboost_acc = accuracy_score(y_val, xgboost_pred)
lgb_acc = accuracy_score(y_val, lgb_pred)
```

Figure14

```
st.write("**Safety Conditions:**")
st.write(safety_conditions[attack_type])

def output(data_path, model_filepath, attack_type):
    try:
        with open(model_filepath, 'rb') as file:
            data = pd.read_csv(data_path)
            model = pickle.load(file)
            try:
                pred = model.predict(data)
            except:
                # pred = np.asscalar(data[data['level']==attack_type]['prob'].values)
                pred = np.ndarray.item(data[data['level']==attack_type]['prob'].values)

    except:
        data = pd.read_csv(data_path)
        # pred = np.asscalar(data[data['level']==attack_type]['prob'].values)
        pred = np.ndarray.item(data[data['level']==attack_type]['prob'].values)
    return pred

probability_score = output(data_path, model_filepath, attack_type)
```

Figure15

3 Steps to Run and execute the codes

Step 1: Initially the user has to go to the google drive which contain the codes and data.

Step 2: Run the colab file.

Step 3: Access authentication for the drive must be made.

Step 4: Now record the application by start running it in the Anaconda Prompt.

Step 5: Run the code python -m streamlit run app. Py