National College of Ireland

Configuration Manual

MSc Internship
MSc Cybersecurity

Suhas Jagannath
Student ID: x19113781

School of Computing
National College of Ireland

Supervisor: Mr Vikas Sahni

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Suhas Jagannath		
Student ID:	X19113781		
Programme:	MSc Cybersecurity	**Year:**	2020
Module:	Academic Internship		
Lecturer:	Mr Vikas Sahni		
Submission Due Date:	17/08/2020		
Project Title:	IoT botnet detection using machine learning techniques		
Word Count:	1411	**Page Count:**	8

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.
I agree to an electronic copy of my thesis being made publicly available on NORMA the National College of Ireland's Institutional Repository for consultation.

Signature: Suhas Jagannath

Date: 17/08/2020

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

Suhas Jagannath
x19113781
MSc Cybersecurity

1 Introduction

The configuration manual briefly explains the hardware and software requirements used for the implementation of the research project. The Configuration manual also consists of python code used in the development of the research project in detail: ***"IoT botnet detection using machine learning techniques."***

2 System Configuration

2.1 Hardware Requirement

A system with the following hardware specification was used for the implementation of the project:

1. ***Processor:*** Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz, 1800 Mhz, 4 Core(s), 8 Logical Processor(s)
2. ***RAM:*** 8 GB
3. ***Operating System:*** MS Windows 10, 64-bits
4. ***GPU:*** Intel(R) UHD Graphics Family, 4GB
5. ***Storage:*** 1TB HDD and 256 SSD

2.2 Software Requirement

The following software was used in the implementation of the research project

1. ***Microsoft Excel:*** This spreadsheet tool was used in every step of this project from data collection, reading from different data sources, and visualizing the results.
2. ***Anaconda Navigator-Jupyter Notebook:*** Anaconda distribution of python is included in Anaconda Navigator. You can build and edit documents with the Jupyter Notebook application that represents the input and output of a Python or R language script. For the faster execution of the program, we moved to Google Colab.
3. ***Google Colaboratory:*** A free cloud-service platform offered by Google to run machine learning models in an environment similar to Jupyter notebook. The GPU, TPU hardware accelerator settings are provided. For high-level machine learning models, efficient GPU is required. Hence, Colab provides Tesla K80 of 12GB GPU which eliminates "out of Memory" warning.

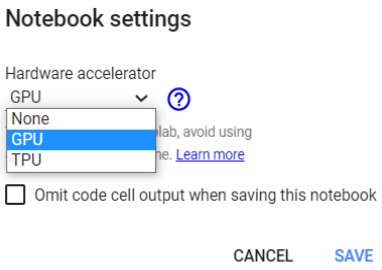

Figure 1: Google Colaboratory Notebook Settings

3 Project Development

Google Colab and Python were used to for the entire implementation of the IoT botnet predictive models. The Implementation consists of Dataset preparation, Feature Engineering, and Predictive Analysis using various machine learning and deep learning techniques.

3.1 Data Preparation

Various libraries such as Pandas and NumPy were used for importing, reading, and manipulation of the data.

For this research, the dataset was prepared by combining both *UNSW_NB15_training-set.csv* and *UNSW_NB15_testing-set.csv* with the help of the Concat function. The final dataset "*finaldataset.csv"* with 257673 rows x 45 columns was transformed and principle component analysis (PCA) was performed to understand the variance-covariance structure of a set of variables through linear combinations.

```
frames = [testing_set_df, training_set_df]
```

```
finaldataset_df= pd.concat(frames)
```

```
finaldataset_df
```

Figure 1: Merging of datasets.

```
IoT_data = pd.read_csv(r'C:\Users\suhas\OneDrive\Documents\thesis\filedataset.csv')

# Machine Learning systems work with integers, we need to encode these
# string characters into ints

encoder = LabelEncoder()

# Now apply the transformation to all the columns:
for col in IoT_data.columns:
    IoT_data[col] = encoder.fit_transform(IoT_data[col])

X_features = IoT_data.iloc[:,1:45]
y_label = IoT_data.iloc[:, 0]
```

```
# Scale the features
scaler = StandardScaler()
X_features = scaler.fit_transform(X_features)
```

Figure 2: Transformation and feature scaling of the dataset.

```
# Visualize
pca = PCA()
pca.fit_transform(X_features)
pca_variance = pca.explained_variance_

plt.figure(figsize=(8, 6))
plt.bar(range(44), pca_variance, alpha=0.5, align='center', label='individual variance')
plt.legend()
plt.ylabel('Variance ratio')
plt.xlabel('Principal components')
plt.show()
```

Figure 3: Principle Component Analysis

3.2 Feature Engineering

Since the final dataset had 45 features that make the predictive models slower. Feature selection was employed using the spearman's correlation thereby achieving dimensionality reduction without losing out important features.

```
#feature extraction using correlation matrix
data = r'C:\Users\suhas\OneDrive\Desktop\correl_data.csv'
df = pd.read_csv(data)
df.corr()
```

Figure 4: Generating the correlation matrix

```
#spearman correlation technique
df.corr(method='spearman')
corrmat = df.corr()
```

Figure 5: Spearman Correlation Technique

Tree-based feature selection also was implemented to compare the results with the Spearman's correlation technique. The below code was implemented to achieve a tree-based feature selection.

```
# Tree-based feature selection:
from sklearn.feature_selection import SelectFromModel
import sklearn.ensemble as ske
fsel = ske.ExtraTreesClassifier().fit(X, y)
model = SelectFromModel(fsel, prefit=True)
X_new = model.transform(X)
nb_features = X_new.shape[1]
indices = np.argsort(fsel.feature_importances_)[::-1][:nb_features]
for f in range(nb_features):
    print("%d. feature %s (%f)" % (f + 1, iot.columns[2+indices[f]], fsel.feature_importances_[indices[f]]))
features = []
for f in sorted(np.argsort(fsel.feature_importances_)[::-1][:nb_features]):
  features.append(iot.columns[2+f])
```

Figure 6: Tree-based feature selection

4 Predictive Models

Modeling for this research was carried out using the python scikit libraries. Random forest, AdaBoost, Decision tree, KNN, Keras, and TensorFlow libraries were also used in predictive modeling. For evaluation of the different predictive model libraries such as metrics, ROC, ROC-AUC scores were used, and also matplotlib library was used for evaluation purposes.

4.1 Data Split

Before implementing the model, we need to read the data with the selected attributes and later the dataset was split into 70% training data & 30% testing data using the following code.

```
from sklearn.model_selection import train_test_split
# implementing train-test-split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=66)
```

Figure 7: Code of data split used for Machine learning models

```
X_train, X_val_and_test, y_train, y_val_and_test = train_test_split(X_scale, y, test_size=0.3)

X_val, X_test, y_val, y_test = train_test_split(X_val_and_test, y_val_and_test, test_size=0.5)

print(X_train.shape, X_val.shape, X_test.shape, y_train.shape, y_val.shape, y_test.shape)
```

Figure 8: Code of data split used for Deep Learning model

```
# Load scikit's random forest classifier library
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Load pandas
import pandas as pd

# Load numpy
import numpy as np

# Set random seed
np.random.seed(0)
```

```
iot = pd.read_csv('/content/drive/My Drive/testing_data.csv')
```

Figure 9: Code snippet for loading dataset

4.2 Random Forest Predictive Model

The code used for training the machine learning model using the random forest algorithm is shown in the below figure. The below code creates the random forest model which predicts the IoT botnet from the test dataset. The test accuracy of the model along with train accuracy is calculated using the code and also the time required to train the model and test the model is also calculated using the below code. The below code also shows the use of hyperparameters for predictive analysis.

```
from sklearn import metrics
from sklearn import model_selection
import time
# random forest model creation
t0 = time.time()
rfc = RandomForestClassifier(n_estimators=200, max_features=8, max_depth= 15 ,random_state=0, min_samples_split = 2, min_samples_leaf = 1,  criterion='gini', oob_score = True)
rfc.fit(X_train,y_train)
t1 = time.time()
print("Model Training time:", (t1 - t0))
# predictions
rfc_predict_test = rfc.predict(X_test)
rfc_predict_train = rfc.predict(X_train)
score_rfc_train = metrics.accuracy_score(y_train, rfc_predict_train)
score_rfc_test = metrics.accuracy_score(y_test, rfc_predict_test)
t2 = time.time()
print("Model Detection time:", (t2 - t1))
print("Accuracy of random forest on train data :",score_rfc_train)
print("Accuracy of random forest on test data :",score_rfc_test)
```

Figure 10: Random Forest Predictive model

The errors such as mean absolute error (MAE), mean squared error (MSE), and root mean squared error (RMSE) is calculated for the random forest predictive model using the below code.

```
# errors in testing data
from sklearn import metrics
print('Mean Absolute Error(testing):', metrics.mean_absolute_error(y_test, rfc_predict_test))
print('Mean Squared Error(testing):', metrics.mean_squared_error(y_test, rfc_predict_test))
print('Root Mean Squared Error(testing):', np.sqrt(metrics.mean_squared_error(y_test, rfc_predict_test)))



# errors in training data
print('Mean Absolute Error(training):', metrics.mean_absolute_error(y_train, rfc_predict_train))
print('Mean Squared Error(training):', metrics.mean_squared_error(y_train, rfc_predict_train))
print('Root Mean Squared Error(training):', np.sqrt(metrics.mean_squared_error(y_train, rfc_predict_train)))
```

Figure 11: Total errors in the Random Forest model

4.3 AdaBoost Predictive Model

The AdaBoost predict model was developed using the following code where the n_estimators of 100 and n_depth of 5 were considered. This boosting algorithm performs training on 70 % of the dataset and predicts the IoT botnet using 30% of the dataset.

```
from sklearn import metrics
from sklearn import model_selection
from sklearn.tree import DecisionTreeClassifier
import time
# Adaboost model creation
t0=time.time()
ada = AdaBoostClassifier(DecisionTreeClassifier(max_depth=5),n_estimators=100)
ada.fit(X_train,y_train)
t1=time.time()
print("Model Training time:", (t1 - t0))
# predictions
ada_predict_test = ada.predict(X_test)
ada_predict_train = ada.predict(X_train)
score_ada_train = round(metrics.accuracy_score(y_train, ada_predict_train) * 100, 2)
score_ada_test = round(metrics.accuracy_score(y_test, ada_predict_test) * 100, 2)
t2=time.time()
print("Model Detection time:", (t2 - t1))
print("Accuracy of adaboost on train data :",score_ada_train)
print("Accuracy of adaboost on test data :",score_ada_test)
```

Figure 12: AdaBoost Predictive model

The errors such as mean absolute error (MAE), mean squared error (MSE), and root mean squared error (RMSE) is calculated for the AdaBoost predictive model using the below code.

```
# errors in testing data
from sklearn import metrics
print('Mean Absolute Error(testing):', metrics.mean_absolute_error(y_test, ada_predict_test))
print('Mean Squared Error(testing):', metrics.mean_squared_error(y_test, ada_predict_test))
print('Root Mean Squared Error(testing):', np.sqrt(metrics.mean_squared_error(y_test, ada_predict_test)))
# errors in training data
print('Mean Absolute Error(training):', metrics.mean_absolute_error(y_train, ada_predict_train))
print('Mean Squared Error(training):', metrics.mean_squared_error(y_train, ada_predict_train))
print('Root Mean Squared Error(training):', np.sqrt(metrics.mean_squared_error(y_train, ada_predict_train)))
```

Figure 13:Errors associated with the AdaBoost predictive model

4.4 KNN Predictive Model

The KNN predict model with n_neighbors of 40 was implemented along with training time and testing time using the below code.

```
from sklearn import metrics
from sklearn import model_selection
from sklearn.neighbors import KNeighborsClassifier
import time
# KNN model creation
t0 = time.time()
knn = KNeighborsClassifier(n_neighbors=40)
knn.fit(X_train,y_train)
t1 = time.time()
print("Model Training time:", (t1 - t0))
# predictions
knn_predict_test = knn.predict(X_test)
knn_predict_train = knn.predict(X_train)
score_knn_train = round(metrics.accuracy_score(y_train, knn_predict_train) * 100, 2)
score_knn_test = round(metrics.accuracy_score(y_test, knn_predict_test) * 100, 2)
t2 =time.time()
print("Model Detection time:", (t2 - t1))
print("Accuracy of KNN on train data :",score_knn_train)
print("Accuracy of KNN on test data :",score_knn_test)
```

Figure 14: KNN Predictive model

The below code was implemented to calculate the errors for the K values between 1 and 40

```
error = []

# Calculating error for K values between 1 and 40
for i in range(1, 40):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train, y_train)
    knn_predict_test = knn.predict(X_test)
    error.append(np.mean(knn_predict_test != y_test))
```

Figure 15: Code for calculating the errors

```
print(error)
plt.figure(figsize=(12, 6))
plt.plot(range(1, 40), error, color='red', linestyle='dashed', marker='o',
        markerfacecolor='blue', markersize=10)
plt.title('Error Rate K Value')
plt.xlabel('K Value')
plt.ylabel('Mean Error')
```

Figure 16: Code for plotting the errors

4.5 Dense Neural Network

For this research, we have implemented 3 dense layers with 2 layers having 'relu' activation and 1 layer having 'sigmoid' activation. The following code was used for the deep learning predictive model.

```
model = Sequential([Dense(32, activation='relu', input_shape=(17,)), Dense(32, activation='relu'), Dense(1, activation='sigmoid'),])

model.summary()

model.layers

model.compile(optimizer='sgd', loss='binary_crossentropy', metrics=['accuracy'])

history = model.fit(X_train, y_train, batch_size=32, epochs=50)

history = model.fit(X_train, y_train,validation_data = (X_test,y_test), epochs=50, batch_size=32)

model.evaluate(X_test, y_test)[1]
```

Figure 17: Dense Neural Predictive model

4.6 Evaluation

We have made use of a confusion matrix to evaluate the accuracy, precision, recall, and F1-score. We have also calculated the ROC-AUC score for each machine learning models. For the deep learning model, model accuracy and loss associated with the model were calculated.

- Classification report and Confusion Matrix:

```
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report, confusion_matrix

rfc_cv_score = cross_val_score(rfc, X, y, cv=10, scoring='roc_auc')


print("=== Confusion Matrix ===")
print(confusion_matrix(y_test, rfc_predict_test))
print('\n')
print("=== Classification Report ===")
print(classification_report(y_test, rfc_predict_test))
print('\n')
print("=== All AUC Scores ===")
print(rfc_cv_score)
print('\n')
print("=== Mean AUC Score ===")
print("Mean AUC Score - Random Forest: ", rfc_cv_score.mean())
```

Figure 18: code for classification report & confusion matrix

- ROC-AUC Scores:

```
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
# calculate roc curve
fpr_rf, tpr_rf, thresholds = roc_curve(y_test, rfc_predict_test)
# calculate AUC
auc = roc_auc_score(y_test, rfc_predict_test)
print('AUC: %.3f' % auc)
```

Figure 19: Code for the ROC-AUC scores

- ROC curve plot:

```
import matplotlib.pyplot as plt
from sklearn import metrics
# fpr means false-positive-rate
# tpr means true-positive-rate
fpr_rf, tpr_rf, _ = metrics.roc_curve(y_test,rfc_predict_test )
auc_score = metrics.auc(fpr_rf, tpr_rf)
# clear current figure
plt.clf()
plt.title('ROC Curve for RandomForest Model')
plt.plot(fpr_rf, tpr_rf, label='AUC = {:.2f}'.format(auc_score))
# it's helpful to add a diagonal to indicate where chance
# scores lie (i.e. just flipping a coin)
plt.plot([0,1],[0,1],'r--')

plt.xlim([-0.1,1.1])
plt.ylim([-0.1,1.1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')

plt.legend(loc='lower right')
plt.show()
```

Figure 20: Plotting ROC curve for the classification models

- Confusion Matrix plot:

```
#Confusion Matrix Visualization
%matplotlib inline
import seaborn as sns
# Get and reshape confusion matrix data
matrix = confusion_matrix(y_test, rfc_predict_test)
matrix = matrix.astype('int') / matrix.sum(axis=1)[:, np.newaxis]

# Build the plot
plt.figure(figsize=(12,5))
sns.set(font_scale=1.4)
sns.heatmap(matrix, annot=True, annot_kws={'size':10},
            cmap=plt.cm.Greens, linewidths=0.2)

# Add labels to the plot
class_names = ['Normal traffic', 'IoT botnet traffic']
tick_marks = np.arange(len(class_names))
tick_marks2 = tick_marks + 0.5
plt.xticks(tick_marks, class_names, rotation=25)
plt.yticks(tick_marks2, class_names, rotation=0)
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.title('Confusion Matrix for Random Forest Model')
plt.show()
```

Figure 21: Confusion Matrix visualization

- Model Accuracy of the Deep Neural Network:

```
import matplotlib.pyplot as plt
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```

Figure 22: Train vs Test accuracy plot

- Model loss of the Deep Neural Network:

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```

Figure 23: Train vs Test loss plot

References

[1]"KNN in Python", *Medium*, 2020. [Online]. Available: https://towardsdatascience.com/knn-in-python-835643e2fb53. [Accessed: 15- Aug- 2020]

[2]"Random Forests Classifiers in Python", *DataCamp Community*, 2020. [Online]. Available: https://www.datacamp.com/community/tutorials/random-forests-classifier-python. [Accessed: 15- Aug- 2020]

[3]"A Guide To Understanding AdaBoost | Paperspace Blog", *Paperspace Blog*, 2020. [Online]. Available: https://blog.paperspace.com/adaboost-optimizer/. [Accessed: 15- Aug- 2020]

[4]"Metrics to Evaluate your Machine Learning Algorithm", *Medium*, 2020. [Online]. Available: https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234. [Accessed: 15- Aug- 2020]

[5]"Classification with TensorFlow and Dense Neural Networks", *Medium*, 2020. [Online]. Available: https://heartbeat.fritz.ai/classification-with-tensorflow-and-dense-neural-networks-8299327a818a. [Accessed: 15- Aug- 2020]

[6]"Hands-on with Feature Selection Techniques: Filter Methods", *Medium*, 2020. [Online]. Available: https://heartbeat.fritz.ai/hands-on-with-feature-selection-techniques-filter-methods-f248e0436ce5. [Accessed: 15- Aug- 2020]