Configuration Manual

MSc Research Project
MSc Cybersecurity

Komal Sharma
Student ID: 20248890

School of Computing
National College of Ireland

Supervisor: Niall Heffernan

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Komal Sharma
Student ID:	20248890
Programme:	MSc Cybersecurity **Year:** 2021-2022
Module:	MSc Research Project
Lecturer:	Niall Heffernan
Submission Due Date:	15-08-2022
Project Title:	Providing Network-Centric Data Security Using Machine Learning and Intrusion Detection
Word Count:	.........1086.................. **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: Komal Sharma

Date: 15-08-2022

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

Providing Network-Centric Data Security Using Machine Learning and Intrusion Detection

Komal Sharma
Student ID: 20248890

1 Introduction

In this Configuration Manual all the essentials required to implement the research and its results on each ensemble-based machine learning modules are mentioned. The software and the hardware requirement along with a screenshot of code for Data Importing and Exploratory Data Analysis (EDA), Data Pre-processing, Label Encoding, Feature Selection, all the 8 different models-accuracy, F1-score, cross validation score and implementation results are included. All 8 different machine Learning models implementation, results and comparison is given in this manual below.

2 System Configuration

In this section the details of Software and Hardware requirements to implement the research is given.

2.1 Hardware Configuration

- Operating System: Windows 10
- System Type: x64-based PC
- Processor: Intel(R) Core(TM) i3-1005G1 CPU @ 1.20GHz, 1190 Mhz, 2 Core(s), 4 Logical Processor(s)
- Hard Disk: 913 GB
- SSD: 119 GB
- RAM: 8.0 GB

2.2 Software Configuration

- Anaconda 3 for Windows
- Jupyter Notebook (Version 6.4.8)
- Python (Version 3.9.12)
- Microsoft Excel

3 Implementation

3.1 Data Collection

The AWID3 dataset is used in this research which is having the data of different types of attacks. I requested for AWID3 dataset from below link and got permission and link to download the dataset. I got approval from professor Dr. Vanessa Ayala related to the ethics form.

https://icsdweb.aegean.gr/awid/download-dataset

The data is divided into different files for each attack containing 50000 data points for each attack, which are merged into one big data.

3.2 Data Exploration

Every Python libraries which are required to implement this research project are listed below in the screenshot of Figure 1.

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.feature_selection import RFE
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier, AdaBoostClassifier, BaggingClassifier
from sklearn.ensemble import GradientBoostingClassifier, StackingClassifier, VotingClassifier, IsolationForest
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.pipeline import make_pipeline
from sklearn.metrics import accuracy_score, f1_score, classification_report, confusion_matrix, make_scorer
from sklearn.model_selection import train_test_split
import warnings
warnings.filterwarnings("ignore")
import matplotlib.pyplot as plt
import seaborn as sns
from numpy import mean
from numpy import std
```

Figure 1: Required Python Libraries

The Figure 2 represents the code to merge the data of different attacks into one big data using pandas concat function.

```
listDF = [deauth, sqlInjection, disas, rouge,botnet,krack, malware, ssh]
data = pd.concat(listDF, ignore_index=True)
```

Figure 2: EDA for Data Merging

The Figure 3 represents the code to check data information and the count of missing values for each feature column.

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400000 entries, 0 to 399999
Columns: 254 entries, frame.encap_type to Label
dtypes: float64(85), int64(22), object(147)
memory usage: 775.1+ MB
```

```
data.isnull().sum()
```

```
frame.encap_type                          0
frame.len                                 0
frame.number                              0
frame.time                                0
frame.time_delta                          0
                                        ...
tls.handshake.session_ticket_length  399797
tls.handshake.version                398828
tls.record.content_type              391111
tls.record.version                   389802
Label                                     0
Length: 254, dtype: int64
```

Figure 3: EDA for Checking Individual Image Size

As seen in Figure 3, 400000 entries, in the data after merging the data and there are 254 columns in the data. The code section also contains the code to print the total number of null values which is the missing data for all the columns.

3.3 Label Encoding

The Figure 4, illustrate the code to encode all the columns of object type.

```
le = LabelEncoder()
```

```
data['Label']= le.fit_transform(data['Label'])
data['frame.time']= le.fit_transform(data['frame.time'])
data['radiotap.present.tsft']= le.fit_transform(data['radiotap.present.tsft'])
data['radiotap.rxflags']= le.fit_transform(data['radiotap.rxflags'])
data['wlan.fc.ds']= le.fit_transform(data['wlan.fc.ds'])
data['wlan.ra']= le.fit_transform(data['wlan.ra'])
data = data.drop(['radiotap.dbm_antsignal'], axis=1)
```

Figure 4: Label Encoding

3.4 Feature Selection

Recursive Feature Estimator is used to select the features using Decision Tree classifier and running a pipeline to find the best features. The Repeated Stratified K-Fold cross-validation is used to check to validate the features selected. Figure 5 below, shows the implementation of this process.

```
rfe = RFE(estimator=DecisionTreeClassifier(), n_features_to_select=5)
```

```
model = DecisionTreeClassifier()
pipeline = Pipeline(steps=[('s',rfe),('m',model)])
```

```
# evaluate model
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
n_scores = cross_val_score(pipeline, X, y, scoring='accuracy', cv=cv, n_jobs=-1, error_score='raise')
```

```
# report performance
print('Accuracy: %.3f (%.3f)' % (np.mean(n_scores), np.std(n_scores)))
```

Figure 5: Feature selection

Figure 6 below, shows the implementation of data splitting. The data is split with 70:30 ratios for train and test set. The figure also shows cross-validation evaluator to check scores of each model and result data frame for store the scores of each model.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=42)
```

```
crossval = make_scorer(f1_score, pos_label = None, average = 'weighted')
```

```
results = pd.DataFrame()
```

Figure 6: Data splitting

3.5 Ensemble Based Machine Learning Models

In this research 8 different Ensemble based ML models are used. The implementation part of these models which is executed in jupyter notebook are given below.

3.5.1 Bagging Classifier

The below screenshot shows the implementation of Bagging Classifier.

```
bc = BaggingClassifier(random_state = 14)
bc.fit(X_train, y_train)
```

```
predBC = bc.predict(X_test)
```

```
accuracy = accuracy_score(y_test, predBC)*100
```

```
f1 = f1_score(y_test, predBC, average='weighted')*100
```

```
print(classification_report(y_test, predBC))
```

```
confusion_matrix(y_test, predBC)
```

```
scores = cross_val_score(bc, X, y, scoring = crossval)
cv = np.round((np.mean(scores) * 100),2)
print("F1: {0:.2f}%".format(cv))
```

```
results= results.append([['Bagging Classifier',accuracy,f1,cv]])
```

3.5.2 AdaBoost Classifier

The below screenshot shows the implementation of Adaboost Classifier.

```
ada = AdaBoostClassifier(random_state = 14)
ada.fit(X_train, y_train)
```

```
predADA = ada.predict(X_test)
```

```
accuracy = accuracy_score(y_test, predADA)*100
```

```
f1 = f1_score(y_test, predADA, average='weighted')*100
```

```
print(classification_report(y_test, predADA))
```

```
confusion_matrix(y_test, predADA)
```

```
scores = cross_val_score(ada, X, y, scoring = crossval)
cv = np.round((np.mean(scores) * 100),2)
print("F1: {0:.2f}%".format(cv))
```

```
results= results.append([['AdaBoost Classifier',accuracy,f1,cv]])
```

3.5.3 Random Forest Trees

The below screenshot shows the implementation of Random Forest Trees.

```
rfc = RandomForestClassifier(random_state = 14)
rfc.fit(X_train, y_train)
```

```
predRFC = rfc.predict(X_test)
```

```
accuracy = accuracy_score(y_test, predRFC)*100
```

```
f1 = f1_score(y_test, predRFC, average='weighted')*100
```

```
print(classification_report(y_test, predRFC))
```

```
confusion_matrix(y_test, predRFC)
```

```
scores = cross_val_score(rfc, X, y, scoring = crossval)
cv = np.round((np.mean(scores) * 100),2)
print("F1: {0:.2f}%".format(cv))
```

```
results= results.append([['RandomForest Classifier',accuracy,f1,cv]])
```

3.5.4 Extra Tree Classifier

The below screenshot shows the implementation of Extra Tree Classifier.

```
etc = ExtraTreesClassifier(random_state = 14)
```

```
etc.fit(X_train, y_train)
```

```
predETC = etc.predict(X_test)
```

```
accuracy = accuracy_score(y_test, predETC)*100
```

```
f1 = f1_score(y_test, predETC, average='weighted')*100
```

```
print(classification_report(y_test, predETC))
```

```
confusion_matrix(y_test, predETC)
```

```
scores = cross_val_score(etc, X, y, scoring = crossval)
cv = np.round((np.mean(scores) * 100),2)
print("F1: {0:.2f}%".format(cv))
```

```
results= results.append([['ExtraTree Classifier',accuracy,f1,cv]])
```

3.5.5 Gradient Boosting Classifier

The below screenshot shows the implementation of Gradient Boosting Classifier.

```
gbc = GradientBoostingClassifier(random_state = 14)
gbc.fit(X_train, y_train)
```

```
predGBC = gbc.predict(X_test)
```

```
accuracy = accuracy_score(y_test, predGBC)*100
```

```
f1 = f1_score(y_test, predGBC, average='weighted')*100
```

```
print(classification_report(y_test, predGBC))
```

```
confusion_matrix(y_test, predGBC)
```

```
scores = cross_val_score(gbc, X, y, scoring = crossval)
cv = np.round((np.mean(scores) * 100),2)
print("F1: {0:.2f}%".format(cv))
```

```
results= results.append([['GradientBoosting Classifier',accuracy,f1,cv]])
```

3.5.6 Isolation Forest Classifier

The below screenshot shows the implementation of Isolation Forest Classifier.

```
ifc = IsolationForest(random_state = 14)
ifc.fit(X_train, y_train)
```

```
predIFC = np.abs(ifc.predict(X_test))
```

```
accuracy = accuracy_score(y_test, predIFC)*100
```

```
f1 = f1_score(y_test, predIFC, average='weighted')*100
```

```
print(classification_report(y_test, predIFC))
```

```
confusion_matrix(y_test, predIFC)
```

```
scores = cross_val_score(ifc, X, y, scoring = crossval)
cv = np.round((np.mean(scores) * 100),2)
print("F1: {0:.2f}%".format(cv))
```

```
results= results.append([['Isolation Forest',accuracy,f1,cv]])
```

3.5.7 Stacking Classifier

The below screenshot shows the implementation of Stacking Classifier.

```
estimators = [('rf', RandomForestClassifier(n_estimators=10, random_state=42)),
              ('svr', make_pipeline(StandardScaler(),
               AdaBoostClassifier(random_state=42)))
             ]
```

```
sc = StackingClassifier( estimators=estimators, final_estimator=BaggingClassifier())
```

```
sc.fit(X_train, y_train)
```

```
predSC = sc.predict(X_test)
```

```
accuracy = accuracy_score(y_test, predSC)*100
```

```
f1 = f1_score(y_test, predSC, average='weighted')*100
```

```
print(classification_report(y_test, predSC))
```

```
confusion_matrix(y_test, predSC)
```

```
scores = cross_val_score(sc, X, y, scoring = crossval)
cv = np.round((np.mean(scores) * 100),2)
print("F1: {0:.2f}%".format(cv))
```

```
results= results.append([['Stacking Classifier',accuracy,f1,cv]])
```

3.5.8 Voting Classifier

The below screenshot shows the implementation of Voting Classifier.

```
vc= VotingClassifier(estimators=[('ada', ada), ('rf', rfc), ('gbc', gbc)], voting='hard')
```

```
vc.fit(X_train, y_train)
```

```
predVC = vc.predict(X_test)
```

```
accuracy = accuracy_score(y_test, predVC)*100
```

```
f1 = f1_score(y_test, predVC, average='weighted')*100
```

```
print(classification_report(y_test, predVC))
```

```
confusion_matrix(y_test, predVC)
```

```
scores = cross_val_score(vc, X, y, scoring = crossval)
cv = np.round((np.mean(scores) * 100),2)
print("F1: {0:.2f}%".format(cv))
```

```
results= results.append([['Voting Classifier',accuracy,f1,cv]])
```

4 Model results

This section explains the performance of the models.

4.1 Models Scores

The below screenshot shows the performance of 8 different ML models in terms of the accuracy, F1-score and cross validation score. Among all of them Random Forest, Extra tree and voting classifier are giving best performance on AWID3 dataset.

```
results.columns= ['Model','Accuracy','F1-Score','CrossValScore']
results
```

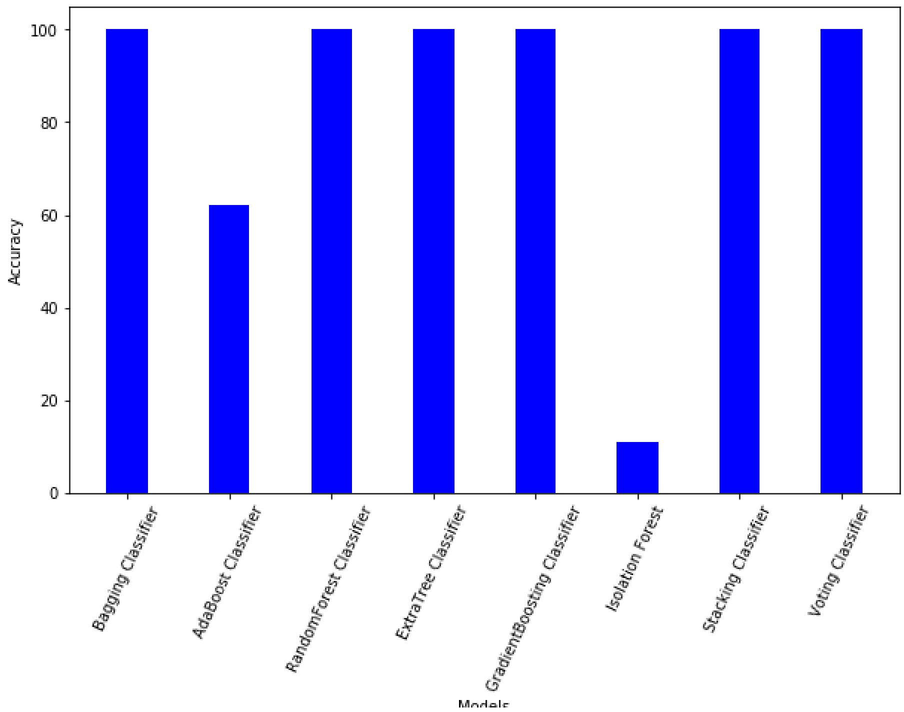
	Model	Accuracy	F1-Score	CrossValScore
0	Bagging Classifier	100.000000	100.000000	98.67
0	AdaBoost Classifier	62.217500	54.730726	41.67
0	RandomForest Classifier	100.000000	100.000000	100.00
0	ExtraTree Classifier	100.000000	100.000000	100.00
0	GradientBoosting Classifier	99.999167	99.999167	100.00
0	Isolation Forest	11.034167	3.224387	9.43
0	Stacking Classifier	99.999167	99.999167	61.38
0	Voting Classifier	100.000000	100.000000	100.00

4.2 Model Accuracy

The below screenshot of the plot of accuracy vs models shows the model accuracy of 8 different ML models.

```
plt.figure(figsize=(10,6))
plt.bar(results['Model'], results['Accuracy'], color ='blue', width = 0.4)
plt.xlabel('Models')
plt.ylabel('Accuracy')
plt.xticks(rotation=65)
```

```
([0, 1, 2, 3, 4, 5, 6, 7], <a list of 8 Text xticklabel objects>)
```

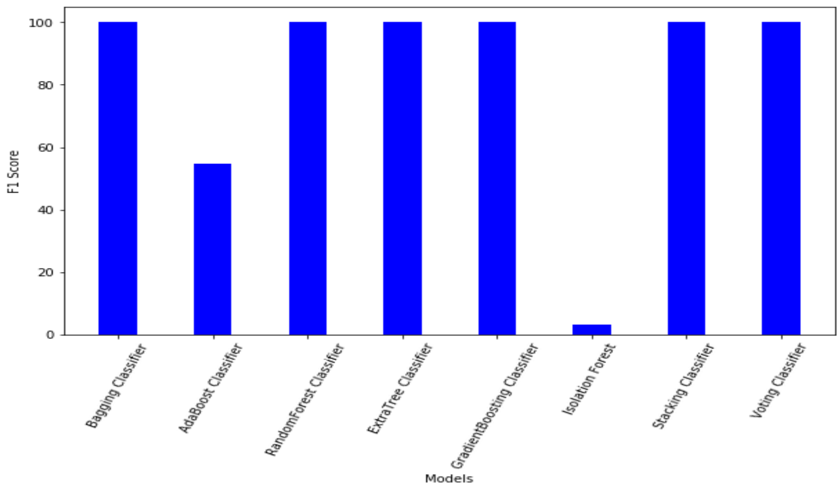


4.3 Model F1-Scores

The below screenshot of the plot of F1-Scores vs models shows the model F1-Scores of 8 different ML models.

```
plt.figure(figsize=(10,6))
plt.bar(results['Model'], results['F1-Score'], color ='blue', width = 0.4)
plt.xlabel('Models')
plt.ylabel('F1 Score')
plt.xticks(rotation=65)
```

```
([0, 1, 2, 3, 4, 5, 6, 7], <a list of 8 Text xticklabel objects>)
```

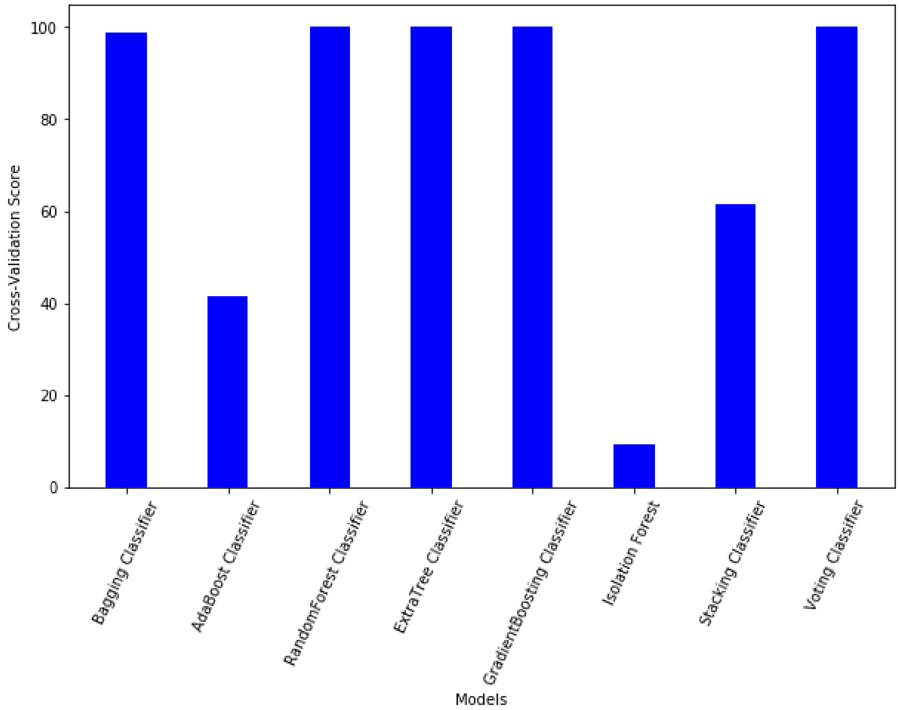


4.4 Model Cross Validation Scores

The below screenshot of the plot of Cross Validation Scores vs models show the model Cross Validation Scores of 8 different ML models.

```
plt.figure(figsize=(10,6))
plt.bar(results['Model'], results['CrossValScore'], color ='blue', width = 0.4)
plt.xlabel('Models')
plt.ylabel('Cross-Validation Score')
plt.xticks(rotation=65)
```

```
([0, 1, 2, 3, 4, 5, 6, 7], <a list of 8 Text xticklabel objects>)
```



4.5 Model Predictions

The below screenshot shows the model prediction of 8 different ML models. In the screenshot we can see that the classifiers prediction values are compared with actual values. If both values are matched that means models can predict the correct types of attacks. Apart from Adboost and Isolation Forest all the Classifiers are able to predict accurately the types of attacks based on AWID3 dataset.

```python
predictions=pd.DataFrame({'Actual': y_test,
                          'Bagging Classifier' : predBC,
                          'RandomForest': predRFC,
                          'AdaBoost': predADA,
                          'ExtraTree': predETC,
                          'GradientBoosting': predGBC,
                          'Isolation Forest' : predIFC,
                          'Stacking': predSC,
                          'Voting': predVC})
```

```python
predictions.head(50)
```

	Actual	Bagging Classifier	RandomForest	AdaBoost	ExtraTree	GradientBoosting	Isolation Forest	Stacking	Voting
23218	1	1	1	2	1	1	1	1	1
20731	1	1	1	2	1	1	1	1	1
39555	1	1	1	2	1	1	1	1	1
147506	2	2	2	2	2	2	-1	2	2
314215	4	4	4	4	4	4	1	4	4
190913	5	5	5	2	5	5	1	5	5
296715	3	3	3	3	3	3	-1	3	3
141482	2	2	2	2	2	2	1	2	2
49119	1	1	1	2	1	1	1	1	1
208005	0	0	0	0	0	0	1	0	0
357337	7	7	7	2	7	7	-1	7	7

References

Ensemble methods. scikit-learn. (2022). from https://scikit-learn.org/stable/modules/ensemble.html.

AWID - Aegean Wi-Fi Intrusion Dataset. Icsdweb.aegean.gr. (2022). from https://icsdweb.aegean.gr/awid/download-dataset.