Configuration Manual

MSc Academic Internship
MSc Cyber Security

Viraj Kudtarkar
Student ID: 18178499

School of Computing
National College of Ireland

Supervisor: Ross Spelman

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Viraj Kudtarkar
Student ID:	18178499
Programme:	MSc Cyber Security **Year:** 2020
Module:	MSc Internship
Supervisor:	Ross Spelman
Submission Due Date:	17/08/2020
Project Title:	Android botnet detection using signature data and Ensemble Machine Learning.
Word Count:	1173 **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.
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: Viraj Kudtarkar

Date: 17 August 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

Viraj Kudtarkar
18178499

1 System Configurations:

OS-MAC Catalina
Processor – Intel Core i7 2.8GHz
Ram 16 Gb
Python – 3.8
IDE – Pycharm 2020.1
Pip 20.2.1

2 Project Libraries:

Database = Sqlite3

Joblib v0.16.0

CSV Reader = pandas v1.1.0

ML = scikit-learn v0.23.2

3 Walkthrough:

3.1 Virtual Environment:

First of all, we setup a virtual environment "Pycharm" IDE to installing all the dependences/Libraries used in our project. We used "Pip3 v20.3.1" package manager for installing different libraries. The most common libraries are pandas, scikit-learn, Joblib and sqlite3.

pandas	1.1.0	1.1.0
pip	20.2.1	▲ 20.2.2
plac	1.1.3	▲ 1.2.0
preshed	2.0.1	▲ 3.0.2
protobuf	3.12.4	▲ 3.13.0
pyasn1	0.4.8	0.4.8
pyasn1-modules	0.2.8	0.2.8
pymongo	3.11.0	3.11.0
pyparsing	2.4.7	2.4.7
python-dateutil	2.7.5	▲ 2.8.1
pytz	2020.1	2020.1
regex	2020.7.14	2020.7.14
requests	2.22.0	▲ 2.24.0
rfc3986	1.4.0	1.4.0
rsa	4.6	4.6
scikit-learn	0.23.2	0.23.2
scipy	1.5.2	1.5.2
selenium	2.53.6	▲ 3.141.0
setuptools	49.2.1	▲ 49.6.0
six	1.12.0	▲ 1.15.0
sniffio	1.1.0	1.1.0
soupsieve	1.9.2	▲ 2.0.1
spacy	2.3.2	2.3.2

Figure 1: Installed Packages

3.2 Dataset:

We used the dataset, for our ML work, which includes the data of botnet and normal applications. Dataset in csv file contains feature set of 15 permissions, 3 intents, family, botnet-category, App Package-Name/MD5 hash keys and labeled column.

Permissions	Intents
android.permission.INTERNET	android.intent.action.BOOT_COMPLETED
android.permission.READ_PHONE_STATE	android.intent.action.POWER_CONNECTED
android.permission.ACCESS_NETWORK_STATE	android.intent.action.BATTERY_LOW
android.permission.WRITE_EXTERNAL_STORAGE	
android.permission.RECEIVE_BOOT_COMPLETED	
android.permission.READ_SMS	
android.permission.SEND_SMS	
android.permission.WRITE_SMS	
android.permission.RECEIVE_SMS	
android.permission.ACCESS_COARSE_LOCATION	
android.permission.ACCESS_FINE_LOCATION	
android.permission.ACCESS_WIFI_STATE	
android.permission.CALL_PHONE	
android.permission.WAKE_LOCK	
android.permission.READ_CONTACTS	

Table 1: Feature set of Permissions and Intents

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
1	android.pern	android.pern	android.pern	android.pern	android.pern	android.pern	android.pern	android.pern	android.pern	android.pern	android.pern	android.pern	android.pern	android.pern	android.pern	android.inter	android.inter	android.inter	MD5	Results
2	1	1	1	1	0	0	0	0	0	1	1	1	0	1	0	0	0	0	1c4e357a8e	1
3	1	1	1	1	0	0	0	0	0	1	1	1	0	1	0	0	0	0	1d15765ffe	1
4	1	1	1	1	0	0	0	0	0	0	0	1	0	1	0	0	0	0	31c657bf77	1
5	1	1	1	1	1	0	0	0	0	1	1	1	0	1	0	0	0	0	37b993b5f5	1
6	1	1	1	1	0	0	0	0	0	0	0	1	0	1	0	0	0	0	3e30f2644a	1
7	1	1	1	1	0	0	0	0	0	1	1	1	0	1	0	0	0	0	41172f215c	1
8	1	1	1	1	1	0	1	0	1	0	0	0	1	1	0	0	0	0	0054d35c7b	1
9	1	1	1	1	1	0	0	0	1	0	0	1	0	1	0	0	0	0	03e8e08061	1
10	1	1	1	1	1	0	1	0	1	0	0	0	1	1	0	0	0	0	1de012d8de	1
11	1	1	1	1	1	0	1	0	1	0	0	0	1	1	0	0	0	0	1e680e1a75	1
12	1	1	1	1	1	0	1	0	1	0	0	0	1	1	0	0	0	0	1e87d0abe9	1
13	1	1	1	1	1	0	1	0	1	0	0	0	1	1	0	0	0	0	2bcf8dd123	1
14	1	1	1	1	1	0	0	0	0	1	1	1	1	1	1	0	0	0	04c12809d3	1
15	1	1	1	1	1	0	0	0	0	1	1	1	1	1	1	0	0	0	40b9b74525	1
16	1	1	1	1	1	0	0	0	0	1	1	1	1	1	1	0	0	0	5d184dc033	1
17	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	a8fe9d0365	1
18	1	1	1	1	0	0	0	0	0	1	0	1	0	1	0	0	0	0	b2bb7d5f5e	1
19	1	1	1	1	0	0	1	0	0	1	0	1	1	1	0	0	0	0	b3c7575f23	1
20	1	1	1	1	0	0	0	0	0	1	0	1	0	1	0	0	0	0	b464a695cf	1
21	1	1	1	1	0	0	0	0	0	1	1	1	0	1	0	0	0	0	b51837d70c	1
22	1	1	1	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	b6d0bb5f7e	1
23	1	1	1	1	1	0	0	0	0	1	1	1	0	1	0	1	0	0	030423f268	1
24	1	1	1	1	1	0	0	0	0	1	1	1	0	1	1	0	0	0	1634b1fb3b	1
25	1	1	1	1	1	0	0	0	0	0	0	1	0	1	0	1	0	0	1652df0226	1
26	1	1	1	1	1	1	1	0	1	1	0	1	0	1	0	1	0	0	7739442c33	1
27	1	1	1	1	1	0	0	0	0	0	0	1	0	1	0	1	0	0	87206282b9	1
28	1	1	1	1	1	0	0	0	0	1	1	1	0	1	0	1	0	0	9324376e27	1
29	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	0	0	a3b05085ea	1
30	1	1	1	1	1	0	0	0	0	1	1	1	0	1	0	1	0	0	a4fcaf1992d	1
31	1	1	1	1	1	0	0	0	0	1	1	1	0	1	0	0	0	0	0c67d0919e	1
32	1	1	1	1	1	0	0	0	0	1	1	1	0	1	0	1	1	0	162cb09e2e	1
33	1	1	1	1	1	1	1	1	1	1	1	1	0	1	0	1	1	0	1be29a6622	1
34	1	1	1	1	1	0	0	0	0	1	1	1	0	1	0	0	0	0	6a66635b6b	1
35	1	1	1	1	1	0	0	0	0	1	1	1	0	1	0	1	1	0	6fc29ab75d	1
36	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	0	7cd86d83d9	1

Figure 2: Dataset CSV

3.3 Database:

We used a sqlite database to store the android application's unique identifier "Package-name" or MD5 hash key-value. Database helped us in making our system more efficient by returning responses in very less time. Database contains a table named "Apps" which stores all the app's information. We have data of almost 600 Normal and botnet applications in a json format. This json format data was inserted in our database for later use.

```python
def create_connect_db():
    try:
        sqlite_connection = sqlite3.connect('SQLite_Python.db')
        return sqlite_connection

    except sqlite3.Error as error:
        print("Error while connecting to sqlite", error)


def create_table():
    db_conn = create_connect_db()
    cursor = db_conn.cursor()
    # Create a table.
    query = """CREATE TABLE IF NOT EXISTS Apps (ID INTEGER PRIMARY KEY, PACKAGE_NAME VARCHAR(100), IS_BOTNET INTEGER); """
    cursor.execute(query)
    db_conn.commit()
    cursor.close()


# Returns true if table already contains apps data else returns false
def table_contain_data():
    db_conn = create_connect_db()
    query = "SELECT * FROM Apps"
    cursor = db_conn.execute(query)
    data = cursor.fetchall()
    cursor.close()
    db_conn.close()

    if len(data) > 0:
```

Figure 3: Database and Table Creation

```python

# method checks if app does not exists in db then returns 0,
# if exists and app is normal then -1,
# if exists and app is botnet then 1.
def check_app_db(package_name):
    db_conn = create_connect_db()
    query = "SELECT * FROM Apps WHERE PACKAGE_NAME LIKE '" + package_name + "'"
    cursor = db_conn.execute(query)
    data = cursor.fetchall()
    cursor.close()
    db_conn.close()
    if len(data) > 0:
        return data[0][2]
    else:
        return 0

```

Figure 4: Querying Database

3.4 Data Pre-processing:

We selected 18 total features (permission and intent), which are most important in distinguishing botnet applications, for training our ML models. In preprocessing part, we removed the noise from data by deleting rows which are incomplete. Also, we dropped the columns that are not necessary for our training process which are "family", "category" and "MD5". After filtering process, data was spitted into 2 parts with 70:30 ratio, 70% for training and 30% for testing.

```python

def get_dataset():
    dataset = pd.read_csv("datasets/data.csv")
    # Preprocess Data (removing extra columns)
    dataset = dataset.drop('family', 1)
    dataset = dataset.drop('category', 1)
    dataset = dataset.drop('MD5', 1)
    x = dataset.iloc[ : , :-1].values
    y = dataset.iloc[:, -1:].values
    x_train, x_test, y_train, y_test = train_test_split(x,y,test_size = 0.30, random_state =1)
    return x_train,x_test,y_train,y_test

```

Figure 5: Dataset preprocessing and splitting

3.5 Classifier Training:

We have trained 5 different classifiers which includes: Logistic Regression, Random Forest, Support Vector Machine, Decision Tree and Naive Bayes, for our machine learning predictor. Models were trained on 70% of dataset. In training process, first the model is created and trained by "model.fit()" method. "Model.predict()" is used for getting test results out of the model and returns a confusion matrix which is used for finding the accuracy of the trained model. We extracted 3 different performance measures for our models which are Accuracy, Precision and Recall. At last, after training process, all trained models were stored in "pkl" file format.

Source code for training the Logistic Regression in mentioned in figure 6.

```python
def train_lr():
    data_train, data_test, label_train, label_test = get_dataset()
    model = LogisticRegression(random_state=0)
    model.fit(data_train, label_train.ravel())

    # predicting the tests set result
    pred = model.predict(data_test)
    create_confusion_matrix(label_test, pred)

    # storing trained models
    store_model(model, "trained_models/lr.pkl")
```

Figure 6: Logistic Regression Model

Figure 7 contains the code for training Random forest Model with best and appropriate parameters.

```python
def train_rf():
    data_train, data_test, label_train, label_test = get_dataset()
    model = RandomForestClassifier(n_estimators=100, criterion="gini", max_features='log2', random_state=0)
    model.fit(data_train, label_train.ravel())

    # predicting the tests set result
    pred = model.predict(data_test)
    create_confusion_matrix(label_test, pred)

    # storing trained models
    store_model(model, "trained_models/rf.pkl")
```

Figure 7: Random Forest Model

Source code for training the SVM in mentioned in figure 6.

```python
def train_svm():
    data_train, data_test, label_train, label_test = get_dataset()
    model = SVC(C=1000, kernel='rbf', gamma=0.2, random_state=0)
    # classifier.fit(x_train, y_train.ravel())
    model.fit(data_train, label_train.ravel())

    # predicting the tests set result
    pred = model.predict(data_test)
    create_confusion_matrix(label_test, pred)

    # storing trained models
    store_model(model, "trained_models/svm.pkl")
```

Figure 8: Support Vector Machine Model

Following figure 9 contains the code of Naïve Bayes model training and storing in the "pkl" file format.

```python
# Naive Bayes
def train_nb():
    data_train, data_test, label_train, label_test = get_dataset()
    model = GaussianNB()
    # classifier.fit(x_train, y_train.ravel())
    model.fit(data_train, label_train.ravel())

    # predicting the tests set result
    pred = model.predict(data_test)
    create_confusion_matrix(label_test, pred)

    # storing trained models
    store_model(model, "trained_models/nb.pkl")
```

Figure 9: Naive Bayes Model

Source code for training the Decision Tree in mentioned in figure 10.

```python
def train_dt():
    data_train, data_test, label_train, label_test = get_dataset()
    model = DecisionTreeClassifier(random_state=0)
    # classifier.fit(x_train, y_train.ravel())
    model.fit(data_train, label_train.ravel())

    # predicting the tests set result
    pred = model.predict(data_test)
    create_confusion_matrix(label_test, pred)

    # storing trained models
    store_model(model, "trained_models/dt.pkl")
```

Figure 10: Decision Tree Model

Figure 11 is the code for creating confusion matrix , calculating accuracy, precision and recall values.

```python
def create_confusion_matrix(testset, predictions):
    # confusion matrix
    cm = confusion_matrix(testset, predictions)
    print("Coufusion Matrix: ", cm)

    # Accuracy
    accuracy = (cm[0][0] + cm[1][1]) / (cm[0][0] + cm[0][1] + cm[1][0] + cm[1][1])
    print("Accuracy:", accuracy)

    # Precision
    precision = cm[0][0]/(cm[0][0]+cm[1][0])
    print("Precision:", precision)

    # Recall
    recall = cm[0][0] / (cm[0][0] + cm[0][1])
    print("Recall:", recall)
```

Figure 11: Confusion Matrix Creation

3.6 Confusion Matrix

Data trained to the five above mentioned algorithms is analyzed during the training session with predicted results against the actual results.

1. **Logistic Regression**

The predicted true and predicted false results for the Logistic regression algorithm in comparison with actual true and actual false values are given in Table 2 below.

Logistic regression	Predicted True	Predicted False
Actual True	328	10
Actual False	45	61

Table 2: LR Confusion Matrix

The accuracy, precision and recall values on the basis of recorded results for Logistic regression has been identified as given in Table 3 below.

Logistic Regression		
1	Accuracy	87.61%
2	Precision	87.93%

3	Recall	97.04%

Table 3

2. Support Vector Machine (SVM)

The predicted results for the Support Vector Machine (SVM) algorithm in comparison with actual true and actual false values are given in Table 4 below.

SVM	Predicted True	Predicted False
Actual True	336	2
Actual False	18	88

Table 4: SVM Confusion Matrix

The recorded accuracy, precision and recall values for Support Vector Machine (SVM) algorithm is recorded as given in Table 5 below.

Support Vector Machine (SVM)		
1	Accuracy	95.49%
2	Precision	94.91%
3	Recall	99.40%

Table 5

3. Random Forest

The predicted results for the Random Forest algorithm with actual and predicted values as recorded in Table 6 given below.

Random Forest	Predicted True	Predicted False
Actual True	332	6
Actual False	19	87

Table 6: RF Confusion Matrix

The calculated values of precision, accuracy and recall have been decided on the basis of the total number of predicted (True, False) and the total number of actual (True, False) values given in table 7 below.

Random Forest		
1	Accuracy	94.36%
2	Precision	94.58%
3	Recall	98.22%

Table 7

4. Decision Tree

Results table recording the predicted and actually true, false values for the Decision Tree algorithm during the testing phase is shown in Table 8 below.

Decision Tree	Predicted True	Predicted False
Actual True	333	5
Actual False	18	88

Table 8: DT Confusion Matrix

Accuracy, precision and recall values calculated on the basis of results recorded in table 9 are listed below in table 9.

Decision Tree		
1	Accuracy	94.81%
2	Precision	94.87%
3	Recall	98.22%

Table 9

5. Naïve Bayes

Training datasets to the Naïve Bayes algorithm are further tested on the basis of 20% of the overall dataset and the results (true, false) have been recorded as listed in table 10 below.

Naïve Bayes	Predicted True	Predicted False
Actual True	624	16
Actual False	61	45

Table 10: NB Confusion Matrix

The accuracy, precision and recall values recorded from the predicted and actual result calculations are given in Table 11 below.

Naïve Bayes		
1	Accuracy	83.10%
2	Precision	84.15%
3	Recall	95.85%

Table 11

3.7 Analysis and Model selection

The recorded results are compared based on the values of the overall percentage of accuracy, precision and recall. The top three models are then further shortlisted with higher accuracy values i.e., Support Vector Machine (SVM), Decision Tree and Random Forest.
Table 1.10 given below gives a better understanding of the top 3 selected models.

ID	Parameters	SVM	Random Forest	Decision Tree
1	Accuracy	95.49%	94.36%	94.81%
2	Precision	94.91%	94.58%	94.87%
3	Recall	99.40%	98.22%	98.22%

3.8 Voting-Based Ensemble Method:

We used ensemble methods in our predictor. Ensemble is a technique which combines different models to produce improved results. Ensembles can produce more precise and accurate outputs. We have created voting-based ensemble model which includes 3 models: Random Forest, SVM and Decision Tree. 3 models out of total 5 models were chosen on the basis of accuracy, so only top 3 model with best accuracy results were used in creation of ensembles. Source code written for ensemble is mentioned in figure 12.

```python
classifier_rf = None
classifier_dt = None
classifier_svm = None


# load the pickle file
def load_classifiers():
    global classifier_rf, classifier_dt, classifier_svm
    classifier_rf = joblib.load('models/trained_models/rf.pkl')
    classifier_dt = joblib.load('models/trained_models/dt.pkl')
    classifier_svm = joblib.load('models/trained_models/svm.pkl')


def check_app(vec):
    # checking and predicting
    feature_vec = vec
    prediction_rf = classifier_rf.predict(feature_vec)
    prediction_dt = classifier_dt.predict(feature_vec)
    prediction_svm = classifier_svm.predict(feature_vec)

    # Voting Mechanism
    prediction_result = prediction_rf + prediction_dt + prediction_svm

    if prediction_result > 0:
        return 1
    else:
        return -1
```

Figure 12: Ensemble Source code

References