National College of Ireland

Configuration Manual

MSc Internship
MSc in CyberSecurity

Ashish Ghorpade
Student ID: x18147461

School of Computing
National College of Ireland

Supervisor: Dr. Muhammad Iqbal

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: Ashish Ghorpade

Student ID: x18147461

Programme: MSc in CyberSecurity **Year:** 2019-2020

Module: Internship

Lecturer: Dr. Muhammad Iqbal

Submission Due Date: 8th January 2020

Project Title: Malware classification using Km-SVM

Word Count: 2546 **Page Count:** 16

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: 

Date: 6th January 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

Ashish Ghorpade
Student ID: x18147461

1 Introduction

The research conducted as part of the Industry Internship required the use of malware dataset, automating data pre-processing and evaluation using predictive machine learning models. This configuration manual is designed to assist the user for evaluating, using and tuning the developed project and its code. Environment Setup and Prerequisites provides a step-by-step guide for setting up an environment for the project and the list of prerequisites for replicating the results achieved. Code and Tuning section consist the complete source code developed and the parameters for tuning various parts of the project.

2 Environment Setup and Prerequisites

Machine learning is a resource intensive task. It is crucial that the hardware configuration used can handle such tasks. Following are the minimum system requirements:

- Processor: Intel i7 4th Gen or Intel i5 5th Gen
- Ram: 8GB DDR3
- HDD: 20GB Free Space

The project was developed in python using multiple packages that support machine learning and file operations. Following are the prerequisites:

- Python 3.7.4
- Visual Studio Code
- Python tool for Visual Studio Code
- Cycler 0.10.0
- Hmmlearn 0.2.3
- Joblib 0.14.1
- Kiwisolver 1.1.0
- Matplotlib 3.1.2
- Numpy 1.18.0
- Pyparsing 2.4.6

- python-dateutil 2.8.1
- scikit-learn 0.22
- scipy 1.4.1
- six 1.13.0
- sklearn 0.0

3 Code and Tuning

The code consists of two main files ‘dataset.py’ and ‘hmmmodels.py’. Below is the code. The code is provided with inline and multi-line comments for instructions on tuning and code functionality.

dataset.py

```
# IMPORT BELOW PACKAGES
import os
import numpy as np
from sklearn.model_selection import train_test_split
import pickle

# DICTIONARY FOR ENCODING DATA FROM SEPARATE MALWARE SYSTEM CALL SEQUENCES. ENCODING RANGE 1-120
class Data():
_encode = {
        '01 01':1  ,    '01 02':2  ,    '01 03':3  ,    '01 04':4  ,    '02 01':5  ,
        '02 02':6  ,    '02 03':7  ,    '03 01':8  ,    '03 02':9  ,    '03 03':10 ,
        '03 04':11 ,    '03 05':12 ,    '03 06':13 ,    '03 07':14 ,    '03 08':15 ,
        '03 09':16 ,    '03 0a':17 ,    '03 0b':18 ,    '03 0c':19 ,    '03 0d':20 ,
        '03 0e':21 ,    '04 01':22 ,    '05 01':23 ,    '05 02':24 ,    '05 03':25 ,
        '05 04':26 ,    '05 05':27 ,    '06 01':28 ,    '06 02':29 ,    '06 03':30 ,
        '06 04':31 ,    '06 05':32 ,    '07 01':33 ,    '07 02':34 ,    '08 01':35 ,
        '08 02':36 ,    '08 03':37 ,    '08 04':38 ,    '08 05':39 ,    '08 06':40 ,
        '09 01':41 ,    '09 02':42 ,    '09 03':43 ,    '09 04':44 ,    '09 05':45 ,
        '09 06':46 ,    '09 07':47 ,    '09 08':48 ,    '09 09':49 ,    '0A 01':50 ,
        '0A 02':51 ,    '0A 03':52 ,    '0A 04':53 ,    '0A 05':54 ,    '0A 06':55 ,
        '0A 07':56 ,    '0B 01':57 ,    '0B 02':58 ,    '0B 03':59 ,    '0B 04':60 ,
        '0B 05':61 ,    '0B 06':62 ,    '0B 07':63 ,    '0B 08':64 ,    '0B 09':65 ,
        '0B 10':66 ,    '0B 11':67 ,    '0C 01':68 ,    '0C 02':69 ,    '0D 01':70 ,
        '0D 02':71 ,    '0D 03':72 ,    '0D 04':73 ,    '0D 05':74 ,    '0D 06':75 ,
        '0D 07':76 ,    '0E 01':77 ,    '0E 02':78 ,    '0E 03':79 ,    '0F 01':80 ,
        '0F 02':81 ,    '0F 03':82 ,    '0F 04':83 ,    '0F 05':84 ,    '0F 06':85 ,
        '0F 07':86 ,    '0F 08':87 ,    '10 01':88 ,    '10 02':89 ,    '10 03':90 ,
        '10 04':91 ,    '10 05':92 ,    '11 01':93 ,    '11 02':94 ,    '11 03':95 ,
        '11 04':96 ,    '11 05':97 ,    '12 01':98 ,    '12 02':99 ,    '12 03':100 ,
```

```
        '12 04':101 ,     '12 05':102 ,     '12 06':103 ,     '12 07':104 ,     '12 08':105 ,
        '12 09':106 ,     '12 0a':107 ,     '12 0b':108 ,     '12 0c':109 ,     '12 0d':110 ,
        '13 01':111 ,     '13 02':112 ,     '13 03':113 ,     '13 04':114 ,     '13 05':115 ,
        '13 06':116 ,     '13 07':117 ,     '13 08':118 ,     '13 09':119 ,     '14 01':120 ,
}
def __init__(self, dir = './dataset/'):
        self.dir = dir
        self.encode = {k.upper():v for k,v in self._encode.items()}
        self.labels = []
        self.y = None
        self.X = None

def label_encode(self, label):
        return self.labels.index(label)
def label_decode(self, code):
        return self.labels[code]

def labels_encode(self):
        return [self.label_encode(x) for x in self.labels]

def load(self):
        self.y = []
        self.X = []

        try:
                with open('dataset.cache', 'rb') as fp:
                (self.X, self.y, self.labels) =  pickle.load(fp)
                print(' Load dataset from cache file. OK')
                return self.X, self.y
        except:
                pass

        for cur, sub, _ in os.walk(self.dir):
                self.labels.extend(sub)
                for d in sub:
                for _,__, files in os.walk(cur + '/' + d):
                x_set = []
                y_set = []
                for file in files:
                if '.csv' in file:
                        # HANDLE CSV FILES IF ANY
                        try:
                        data = np.loadtxt(_ + '/' + file, dtype=np.int32, delimiter='\n')
                        x_set.append(np.reshape(data, (-1,1)).tolist())
                        y_set.append(self.label_encode(d))
                        print('File: ' + file + ' OK')
                        except:
                        print('File: ' + file + ' invalid')
```

```
                else:
                        # HANDLE TEXT FILES
                        with open (_ + '/' + file, 'r') as fp:
                        data = [[self.encode[l[:5].upper()]] for l in fp.readlines() if not
                        l.startswith('#')] # filter comment
                        x_set.append(data)
                        y_set.append(self.label_encode(d))
                        print('File: ' + file + ' OK')
                        pass

                self.X.append(x_set)
                self.y.append(y_set)




        with open('dataset.cache', 'wb') as fp:
                pickle.dump((self.X, self.y, self.labels),fp)
                print(' Save dataset from cache file. OK')
        return self.X, self.y

def loadv2(self):
        self.y = []
        self.X = []

        try:
                with open('dataset.cache', 'rb') as fp:
                (self.X, self.y, self.labels) =  pickle.load(fp)
                print(' Load dataset from cache file. OK')
                return self.X, self.y
        except:
                pass

        for cur, sub, _ in os.walk(self.dir):
                self.labels.extend(sub)
                for d in sub:
                for _,__, files in os.walk(cur + '/' + d):
                x_set = []
                y_set = []
                for file in files:
                try:
                        if '.csv' in file:
                        # HANDLE CSV FILES IF ANY
                        try:
                                data = np.loadtxt(_ + '/' + file, dtype=np.int32,
                                delimiter='\n')
                                x_set.append(np.reshape(data, (-1,1)).tolist())
                                y_set.append(self.label_encode(d))
```

```
                        print('File: ' + file + ' OK')
                    except:
                        print('File: ' + file + ' invalid')
                    else:
                    # HANDLE TEXT FILES
                    with open (_ + '/' + file, 'r') as fp:
                        data = [[self.encode[l[:5].upper()]] for l in fp.readlines() if
                        not l.startswith('#')] # filter comment
                        x_set.append(data)
                        y_set.append(self.label_encode(d))
                        print('File: ' + file + ' OK')
                    pass
                except Exception as ex:
                    print('File: ' +  file + ' ' + str(ex))
                print(' Label: ' + d + str(len(y_set)) + str(len(x_set)))
                self.X.extend(x_set)
                self.y.extend(y_set)



        with open('dataset.cache', 'wb') as fp:
                pickle.dump((self.X, self.y, self.labels),fp)
                print(' Save dataset from cache file. OK')
        return self.X, self.y


def split(self, test_size = 0.1, random_state = 42):
        if not self.y and not self.X:
                self.load()

        # SPLIT DATA
        X_train, X_test, y_train, y_test = [], [], [], []
        for X,y  in zip(self.X, self.y):
                X_tr, X_t, y_tr, y_t = train_test_split(X, y, test_size=test_size,
                random_state=42)
                X_train.append(X_tr)
                y_train.append(y_tr)
                X_test.append(X_t)
                y_test.append(y_t)

        return X_train, X_test, y_train, y_test

def splitv2(self, test_size = 0.1, random_state = 42):
        if not self.y and not self.X:
                self.loadv2()
```

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

        return X_train, X_test, y_train, y_test
```

hmmmodels.py

```
from dataset import Data
import numpy as np
from hmmlearn.hmm import GaussianHMM
from sklearn.cluster import KMeans
from sklearn.svm import SVC
import pickle
from scipy.spatial import distance
import matplotlib.pyplot as plt
from sklearn.metrics  import accuracy_score
import time

class HMMClassifier:
   model = {}
   def __init__(self, n_components = 2, n_iter = 800, top_n = 100,  svm_c = 1.0,
svm_gama = 'scale', labels = []):
      try:
         self.read()
         return
      except:
         pass
      self.n_components = n_components
      self.n_iter = n_iter
      self.labels = labels
      self.svm_c = svm_c
      self.svm_gama = svm_gama
      self.top_n = top_n
      self.model={
         'hmm': {},
         'kmean': None,
         'svm': None,
         'labels': labels
      }
      self.build_model()

   def build_model(self):
      # Build HMM
      for i, l in enumerate(self.labels):
         self.model['hmm'][l] =    GaussianHMM(n_components = self.n_components,
n_iter= self.n_iter)
```

```
 #Build Kmean
    self.model['kmean'] = KMeans(n_clusters=len(self.labels))

    #Build svm
    self.model['svm'] = SVC(kernel = 'rbf', gamma=self.svm_gama, C=self.svm_c)

    self.model['kmean_svm'] = SVC(kernel = 'rbf', gamma=self.svm_gama,
C=self.svm_c)

  def train_hmm(self, X, y):
    for i, l in enumerate(labels):
      y_ith = np.array(y)== l
      x_ith = np.array(X)[y_ith].tolist()
      print(' ' * 10 +'Fit HMM: ' + str(l))
      '''try:
        with open('models/HMM_' + str(l) +'.pkl', "rb") as file:
          model = pickle.load(file)
          self.model['hmm'][l] = model
          print(' ' * 15 +'Loading HMM: ' +'models/HMM_' +str(l) +'.pkl')
          continue
      except:
        pass'''
      _x = np.concatenate(x_ith)
      length = [len(x) for x in x_ith]
      self.model['hmm'][l].fit(_x, length)

      '''with open('models/HMM_' +str(l) +'.pkl', "wb") as file:
        pickle.dump(self.model['hmm'][l], file)
        print(' ' * 15 +'Dump HMM: ' +'models/HMM_' +str(l) +'.pkl')'''

  def score_hmm(self, X):
    scores = []
    for x in X:
      score_x = []
      for name, m in self.model['hmm'].items():
        score = m.score(x)/len(x)
        #score = m.score(x)          #Uncomment for same sequence length
        score_x.append(score)
      scores.append(score_x)

    # Vector scores
    X_scores = np.asarray(scores)

    return X_scores

  def cluster_kmean(self, X_score, y = None):
    """
    Y for none use
```

```
    """
    print(' ' * 10 +'Cluster with kmean: ' + str(len(labels)) + ' clusters')
    return self.model['kmean'].fit_predict(X_score)

def kmean_predict(self, X_score):
    return self.model['kmean'].predict(X_score)

def reduce_trainset(self, X_score, y, y_kmean):
    # Find Neighbor
    print(' ' * 10 +'Find  Neighbor set ')
    # Find miss set
    # Store  True label # Algorithm 1 Step 3
    idx_miss  = np.not_equal(y, y_kmean) # Step 4
    X_miss =  np.asarray(X_score[idx_miss])# Step 5

    # Find Neighbor
    # Step 1/2 Algorithm 2
    # Create distance matrix
    n_sample =  len(y)
    print(' ' * 15 +'Create Distance matrix: ' + str(n_sample) + ' x '+str(n_sample))
    distances = np.ndarray(shape= (n_sample,n_sample), dtype = np.float)

    for i in range(n_sample): # Algorithm II, Step 1
        for j in range(n_sample): # Algorithm II, Step 2
            if i != j:
                distances[i][j]= np.linalg.norm(X_score[i]- X_score[j]) # Algorithm II, Step 3
            else:
                distances[i][j] = 0.0
    # Top n-min distance
    print(' ' * 15 +'Iterate for each sample in misC: ')
    nt_set = set() #Init NNeighbor set index step 12
    for i, val in enumerate(idx_miss): #Algorithm I Step 6
        if not val: # Skip same y labe and y cluster
            continue

        #print(' ' *15  + 'Find ' +str(self.top_n) + ' min distance in row: ' + str(i))
        row = distances[i, :]
        # Find top n-min distance
        top_n_min_dis_index = row.argsort()[: self.top_n]
        nt_set.update(set(top_n_min_dis_index))

    nt_index= list(nt_set)
    X_nt = X_score[np.asarray(nt_index)]
    y_nt = np.array(y)[np.asarray(nt_index)]

    return X_nt, y_nt

def train_svm(self, X_nt, y_nt, kmean_svm_selected = True):
```

```
        if kmean_svm_selected:
            start = time.time()
            self.model['kmean_svm'].fit(X_nt, y_nt)
            elapsed_time = (time.time() - start)
            print('Training Time for Kmean+SVM (' + str(len(y_nt))+' samples): ' +
str(elapsed_time))

        else:
            start = time.time()
            self.model['svm'].fit(X_nt, y_nt)
            elapsed_time = (time.time() - start)
            print('Training Time for SVM Only (' + str(len(y_nt))+' samples): ' +
str(elapsed_time))

    def draw_chart(self, y, y_pre, title = "Chart", decode_labels = []):

        result = np.zeros((len(self.labels), len(self.labels))) # nx Cluster x
        for i,v in enumerate(self.labels):
            a_class = y_pre == v
            unique, counts = np.unique(np.array(y)[a_class], return_counts=True)
            result[i][unique] = counts

        b = []
        for i,v in enumerate(self.labels):
            p = plt.bar(self.labels, result[:, v] , 0.5)
            b.append(p)

        plt.title(title)
        plt.xticks(self.labels, tuple(self.labels))
        plt.yticks(np.arange(0, len(y), 100))
        plt.legend(tuple(b), (decode_labels[x] for i,x in enumerate(self.labels) ))

        plt.show()

    def train(self, X,y, plot = True, labels = []):
        self.train_hmm(X, y)
        # Compute Score
        X_score = self.score_hmm(X)
        #Train SVM no kmean
        print(' ' * 10 +'Fit SVM with Gaussian kernel')
        self.train_svm(X_score, y, kmean_svm_selected=False)
        if plot:
            y_pre = self.model['svm'].predict(X_score)
            self.draw_chart(y, y_pre, decode_labels= labels, title= "Train Set - SVM Only")
        # Cluster
        y_kmean = self.cluster_kmean(X_score)

        if plot:
```

```
        self.draw_chart(y, y_kmean, title="Train Set - K mean Only ", decode_labels
=labels)
      X_nt, y_nt = self.reduce_trainset(X_score, y, y_kmean)

      print(' ' * 10 +'NT set (Train set for Kmean+SVM): ' + str(len(y_nt)) + ' samples')
      print(' ' * 10 +'Fit SVM with Gaussian kernel with NT set')
      self.train_svm(X_nt, y_nt, kmean_svm_selected=True)
      if plot:
        y_pre = self.model['kmean_svm'].predict(X_score)
        self.draw_chart(y, y_pre, decode_labels= labels, title= "Train Set -  Kmean &&
SVM")

      self.save()

   def predict(self, X, kmean_svm_selected = True):
      """
       X.shape = [n_sample] sequency
      """
      #print(' ' * 5 +'Predict: ' +  str(len(X)))
      scores = []
      for x in X:
        score_x = []
        for name, m in self.model['hmm'].items():
          score = m.score(x)/len(x)
          #score = m.score(x)                    #Uncomment for same sequence length
          score_x.append(score)
        scores.append(score_x)

      X_score = np.asarray(scores)
      if kmean_svm_selected:
        return X_score, self.model['kmean_svm'].predict(X_score)
      else:
        return X_score, self.model['svm'].predict(X_score)

   def save(self, path = 'models/Model.pkl'):
      with open(path, "wb") as file:
        pickle.dump((self.n_components, self.n_iter, self.labels, self.svm_c,
self.svm_gama, self.top_n, self.model), file)
        print(' ' * 5 +'Save Model: ' + path)

   def read(self, path = 'models/Model.pkl'):
      with open(path, "rb") as file:
        (self.n_components, self.n_iter, self.labels, self.svm_c, self.svm_gama,
self.top_n, self.model) = pickle.load(file)
        print(' ' * 5 +'Loading Model: ' +path)
```

```
#Load data
dataset = Data()
X_train, X_test, y_train, y_test = dataset.splitv2(test_size=0.10) # This is the split the data. Size can be varied.
labels =  dataset.labels_encode()

model = HMMClassifier(n_components=2, n_iter=800, top_n=100, labels = labels)
print('Train Set: ' +  str(len(y_train))  + ' samples') # Iterations can be toggled between 100 - 800 for viewing changes

# Make sure folder 'models' is in the project before running it.
# For training model:
# 1. Delete all files in folder models/*
# 2. Uncomment the line code: #model.train(X_train, y_train, labels=dataset.labels)
# For predicting with existing models (trained)
# 1. Comment out the like code: #model.train(X_train, y_train, labels=dataset.labels)

model.train(X_train, y_train, labels=dataset.labels)

# Testing with data.
print('Test Set: ' +  str(len(y_test))  + ' samples')
print('Accuracy for Test:')

# kmean_svm_selected indicate predict with SVM only (false) or SVM  + Kmean (True)
_, y_pre = model.predict(X_test, kmean_svm_selected= True)
model.draw_chart(y_test,y_pre, title= "Test Set - Kmean && SVM", decode_labels = dataset.labels)
print(' '*5 + 'Kmean - SVM: ' + str(accuracy_score(y_test, y_pre)))

# kmean_svm_selected indicate predict with SVM only (false) or SVM  + Kmean (True)
X_score, y_pre = model.predict(X_test, kmean_svm_selected= False)
model.draw_chart(y_test, y_pre, title= "Test Set - SVM Only", decode_labels = dataset.labels)
print(' '*5 + 'SVM only: ' + str(accuracy_score(y_test, y_pre)))

#Kmean only:
y_pre = model.kmean_predict(X_score)
model.draw_chart(y_test, y_pre, title= "Test Set - Kmean Only", decode_labels = dataset.labels)
print(' '*5 + 'Kmean only: ' + str(accuracy_score(y_test, y_pre)))
```

4 Internship Activity Report

Please find below a signed and scanned copy of the Internship activity report.

Below is the internship activity report. A supporting letter from the Employer has also been attached to this document.

Student Name: Ashish Ghorpade Student number: x18147461

Company: PwC Ireland Duration: 16/09/2019 - 6/12/2019

For the duration of the Internship, I had the opportunity to work as an Information Security Specialist for the Network Information Security (NIS) Team. Majority of my work was on the following:

- Incident response for Phishing and Malware checks through the internal reporting system.
- Endpoint Protection using Cylance Protect, a ML based Anti-Virus and Windows Defender.
- Asset Security and Vulnerability Management using Qualys.
- Application Risk Assessments.

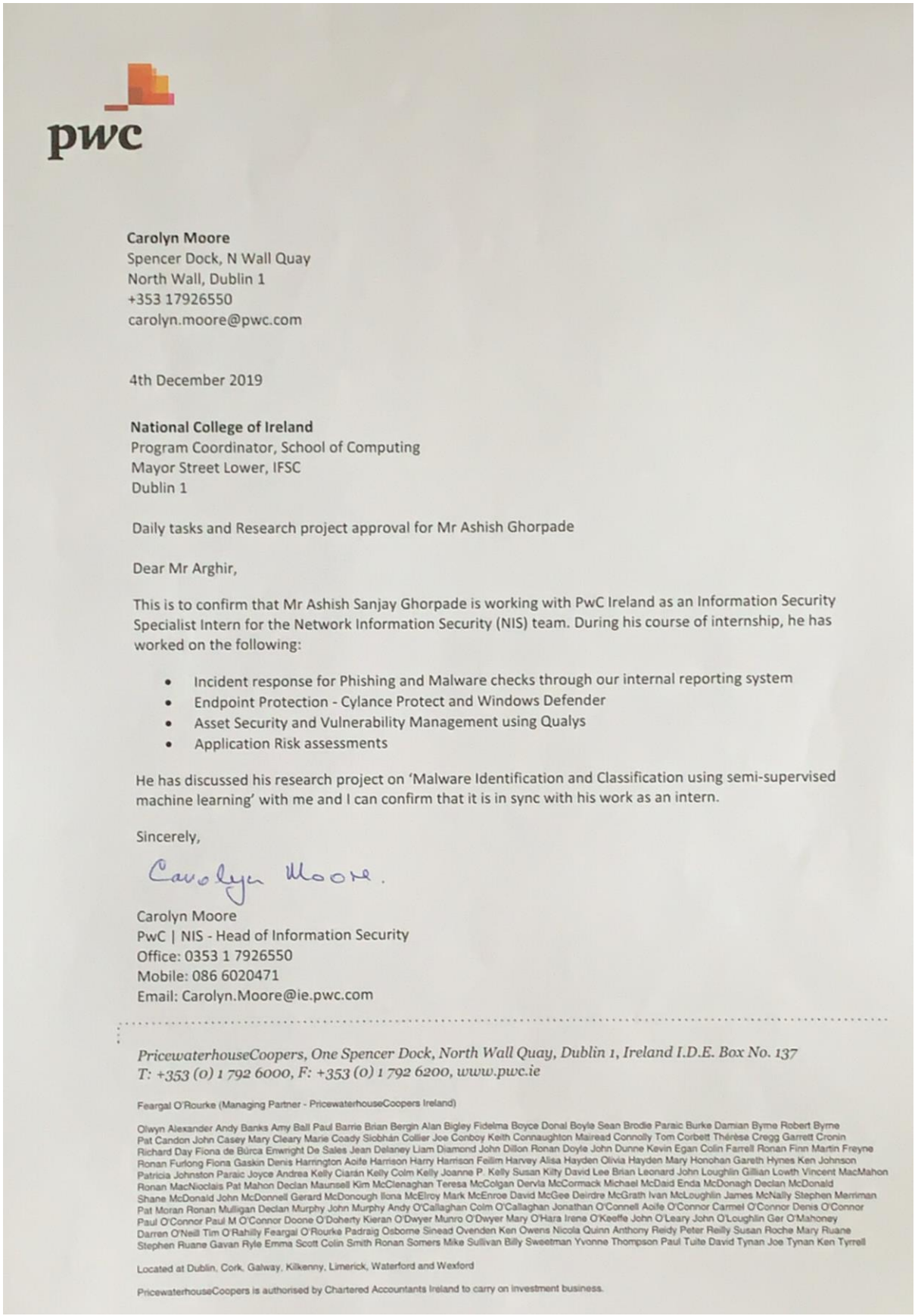
My work on Incident response and Endpoint protection was the basis of selection of my research topic – Malware classification using Machine Learning.

Employer comments

I confirm that Ashish has worked on the above-mentioned tasks for the duration of his internship. His research topic has been discussed and approved.

Student Signature: Ghorpade Date: 06/01/2020

Industry Supervisor Signature: Carolyn Moore. Date: 06/01/2020



Carolyn Moore
Spencer Dock, N Wall Quay
North Wall, Dublin 1
+353 17926550
carolyn.moore@pwc.com

4th December 2019

National College of Ireland
Program Coordinator, School of Computing
Mayor Street Lower, IFSC
Dublin 1

Daily tasks and Research project approval for Mr Ashish Ghorpade

Dear Mr Arghir,

This is to confirm that Mr Ashish Sanjay Ghorpade is working with PwC Ireland as an Information Security Specialist Intern for the Network Information Security (NIS) team. During his course of internship, he has worked on the following:

- Incident response for Phishing and Malware checks through our internal reporting system
- Endpoint Protection - Cylance Protect and Windows Defender
- Asset Security and Vulnerability Management using Qualys
- Application Risk assessments

He has discussed his research project on 'Malware Identification and Classification using semi-supervised machine learning' with me and I can confirm that it is in sync with his work as an intern.

Sincerely,

Carolyn Moore.

Carolyn Moore
PwC | NIS - Head of Information Security
Office: 0353 1 7926550
Mobile: 086 6020471
Email: Carolyn.Moore@ie.pwc.com

References

[1] Rehman, A. (2017). Building Hidden Markov Models. [video] Available at: https://youtu.be/CHxhopCsKYo [Accessed 5 Dec. 2019].

[2] Rehman, A. (2017). Building a Speech Recognizer. [video] Available at: https://youtu.be/i-vO3aTkcAY [Accessed 5 Dec. 2019].

[3] Rehman, A. (2017). Building Hidden Markov Models for Sequential Data. [video] Available at: https://youtu.be/Fc-up710V9A [Accessed 5 Dec. 2019].

[4] Rehman, A. (2017). Analyzing Stock Market Data with Hidden Markov Models. [video] Available at: https://youtu.be/Z_ODtoPelN8 [Accessed 5 Dec. 2019].

[5] Hmmlearn.readthedocs.io. (2010). hmmlearn — hmmlearn 0.2.3 documentation. [online] Available at: https://hmmlearn.readthedocs.io/en/stable/ [Accessed 10 Dec. 2019].