Code Listings

Yue Leng

May 14, 2023

Introduction

The results of this study are presented in the form of 5 different scenarios, and thus codes are structured into the 5 scenarios. Which are:

1. Using decision trees to classify the original datasets.
2. Using decision trees to classify the oversampled datasets.
3. Using DNN to classify the oversampled datasets.
4. Using decision trees to classify the synthetic datasets.
5. Using DNN to classify the synthetic datasets.

For our research, we worked with a total of 23 datasets, and we do data handling and utilize different methods including CTGAN, DNN, Decision Tree and Oversampling methods on various datasets. Thus parts of the code for different datasets should be quite similar except for some tiny modifications such as file paths. In all scenarios except Scenario 4, we combined the procedures for all datasets into a single file. However, in Scenario 4, we used CTGAN to generate synthetic data and applied a decision tree on the synthetic data for each dataset separately. To simplify and clarify the presentation, instead of showing all the code for producing synthetic datasets and getting results for Scenario 4, we only provide one example for Scenario 4. Apart from that, we also utilized analytics for hunting from Threat Hunter Playbook. And for the same reasons, we only give one example of our hunting programs here. Nonetheless, all source code and datasets, including the synthetic datasets, are available on our GitHub repository at: `https://github.com/L-yue-0112/Signature-based-IDS`.

We used Jupyter Notebook for our research. To make it more convenient, we convert part of our code into Python and list them below. However, it is important to note that Jupyter Notebook provides a user-friendly and interactive interface, so we still suggest referring to the original Jupyter Notebook files for optimal display.

Code listings

In what follows, we enumerate all files distributed with this project, for convenience of navigation, we provide an index of all listings below:

Listings

Listing 1: Hunting Program - Example

```
#!/usr/bin/env python
# coding: utf-8

# In[2]:
```

```
import requests
from zipfile import ZipFile
from io import BytesIO

url = 'https://raw.githubusercontent.com/OTRF/Security-Datasets/master/datasets/
    atomic/windows/execution/host/empire_launcher_vbs.zip'
zipFileRequest = requests.get(url)
zipFile = ZipFile(BytesIO(zipFileRequest.content))
datasetJSONPath = zipFile.extract(zipFile.namelist()[0])
```

 # In[3]:

```
import pandas as pd
from pandas.io import json

df = json.read_json(path_or_buf=datasetJSONPath, lines=True)
```

 # In[4]:

```
(
df[['@timestamp','Hostname']]

[((df['Channel'] == 'Windows PowerShell') | (df['Channel'] == 'Microsoft-Windows-
    PowerShell/Operational'))
    & (
      (df['EventID'] == 400)
      | (df['EventID'] == 4103)
    )
]
.head()
)
```

 # In[5]:

```
(
df[['@timestamp','Hostname','NewProcessName','ParentProcessName']]

[(df['Channel'].str.lower() == 'security')
    & (df['EventID'] == 4688)
    & (df['NewProcessName'].str.lower().str.endswith('powershell.exe', na=False))
    & (~df['ParentProcessName'].str.lower().str.endswith('explorer.exe', na=False
       ))
]
.head()
)
```

 # In[6]:

```
(
df[['@timestamp','Hostname','Image','ParentImage']]

[(df['Channel'] == 'Microsoft-Windows-Sysmon/Operational')
    & (df['EventID'] == 1)
```

```
    & (df['Image'].str.lower().str.endswith('powershell.exe', na=False))
    & (~df['ParentImage'].str.lower().str.endswith('explorer.exe', na=False))
]
.head()
)
```

 # In[7]:

```
(
df[['@timestamp','Hostname','Image','ParentImage']]

[(df['Channel'] == 'Microsoft-Windows-Sysmon/Operational')
    & (df['EventID'] == 1)
    & (df['Image'].str.lower().str.endswith('powershell.exe', na=False))
    & (~df['ParentImage'].str.lower().str.endswith('explorer.exe', na=False))
]
.head()
)
```

 # In[8]:

```
(
df[['@timestamp','Hostname','Image','ImageLoaded']]

[(df['Channel'] == 'Microsoft-Windows-Sysmon/Operational')
    & (df['EventID'] == 7)
    & (
        (df['Description'].str.lower() == 'system.management.automation')
        | (df['ImageLoaded'].str.lower().str.contains('.*system.management.
            automation.*', regex=True))
    )
]
.head()
)
```

 # In[9]:

```
(
df[['@timestamp','Hostname','Image','PipeName']]

[(df['Channel'] == 'Microsoft-Windows-Sysmon/Operational')
    & (df['EventID'] == 17)
    & (df['PipeName'].str.lower().str.startswith('\pshost', na=False))
]
.head()
)
```

 # In[10]:

```
(
df[['@timestamp','Hostname','Message']]

[(df['Channel'] == 'Microsoft-Windows-PowerShell/Operational')
```

```
    & (df['EventID'] == 53504)
]
.head()
)


# In[ ]:
```

Listing 2: Scenario 1

```
#!/usr/bin/env python
# coding: utf-8

# # Decision Tree

# ## Datasets

# In[1]:


import os
import pandas as pd
import numpy as np
from pandas.io import json

df = []

datasetJSONPath = os.getcwd() + "\\empire_mimikatz_logonpasswords_2020
    -08-07103224.json"
df1 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

datasetJSONPath = os.getcwd() + "\\
    empire_dllinjection_LoadLibrary_CreateRemoteThread_2020-07-22000048.json"
df2 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

datasetJSONPath = os.getcwd() + "\\
    empire_dcsync_dcerpc_drsuapi_DsGetNCChanges_2020-09-21185829.json"
df3 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

datasetJSONPath = os.getcwd() + "\\
    empire_powerview_ldap_ntsecuritydescriptor_2020-09-21130944.json"
df4 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

datasetJSONPath = os.getcwd() + "\\empire_enable_rdp_2019-05-18203650.json"
df5 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

datasetJSONPath = os.getcwd() + "\\empire_launcher_vbs_2020-09-04160940.json"
df6 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

datasetJSONPath = os.getcwd() + "\\empire_wdigest_downgrade_2019-05-18201922.json
    "
df7 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

datasetJSONPath = os.getcwd() + "\\empire_psremoting_stager_2020-09-20170827.json
    "
df8 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

datasetJSONPath = os.getcwd() + "\\empire_psremoting_stager_2020-09-20170827.json
    "
df9 = json.read_json(path_or_buf=datasetJSONPath, lines=True)
```

```
datasetJSONPath = os.getcwd() + "\\
    empire_mimikatz_backupkeys_dcerpc_smb_lsarpc_2020-10-22143243.json"
df10 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

datasetJSONPath = os.getcwd() + "\\empire_mimikatz_sam_access_2020-09-22041117.
    json"
df11 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

datasetJSONPath = os.getcwd() + "\\empire_mimikatz_sam_access_2020-09-22041117.
    json"
df12 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

datasetJSONPath = os.getcwd() + "\\
    empire_wmi_dcerpc_wmi_IWbemServices_ExecMethod_2020-09-21001437.json"
df13 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

datasetJSONPath = os.getcwd() + "\\
    empire_wmi_local_event_subscriptions_elevated_user_2020-09-04164306.json"
df14 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

datasetJSONPath = os.getcwd() + "\\empire_psinject_PEinjection_2020-08-07143205.
    json"
df15 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

datasetJSONPath = os.getcwd() + "\\empire_psexec_dcerpc_tcp_svcctl_2020
    -09-20121608.json"
df16 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

datasetJSONPath = os.getcwd() + "\\empire_psexec_dcerpc_tcp_svcctl_2020
    -09-20121608.json"
df17 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

datasetJSONPath = os.getcwd() + "\\
    empire_find_localadmin_smb_svcctl_OpenSCManager_2020-09-22021559.json"
df18 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

datasetJSONPath = os.getcwd() + "\\rdp_interactive_taskmanager_lsass_dump_2020
    -09-22043748.json"
df19 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

datasetJSONPath = os.getcwd() + "\\empire_monologue_netntlm_downgrade_2019
    -12-25045202.json"
df20 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

datasetJSONPath = os.getcwd() + "\\msf_record_mic_2020-06-09225055.json"
df21 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

datasetJSONPath = os.getcwd() + "\\
    covenant_wmi_remote_event_subscription_ActiveScriptEventConsumers_2020
    -09-01214330.json"
df22 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

datasetJSONPath = os.getcwd() + "\\covenant_dcom_iertutil_dll_hijack_2020
    -10-09183000.json"
df23 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

datasetJSONPath = os.getcwd() + "\\covenant_wmi_wbemcomn_dll_hijack_2020
    -10-09173318.json"
df24 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

datasetJSONPath = os.getcwd() + "\\covenant_copy_smb_CreateRequest_2020
```

```
    -09-22145302.json"
df25 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

datasetJSONPath = os.getcwd() + "\\
    covenant_lolbin_wuauclt_createremotethread_2020-10-12183248.json"
df26 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

# 5, 7 have code error and 12 doesn't have any result for labelling
df = [df1, df2, df3, df4, df6, df8, df9, df10, df11, df13, df14, df15, df16, df17
    , df18, df19, df20, df21, df22, df23, df24, df25, df26]


# In[39]:


from sklearn import tree
from sklearn import preprocessing
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import metrics

# functions definition:

# convert object items in the list into strings
def preprocess_labelEncoder(df):

    le = preprocessing.LabelEncoder()

    # replace NAN with 0
    df = df.replace(np.nan, 0, regex=True)

    for column in df.columns:
        # if dtype is int, does not need to convert to string , e.g. EventID
        if isinstance(df[column][0], int) and df[column][0] != 0:
            le.fit(df[column])
            df[column] = le.transform(df[column])

        # convert to list of strings and use label encoder to label
        else:
            df[column] = list(map(str, df[column]))
            le.fit(df[column])
            df[column] = le.transform(df[column])


    return df

# draw the decision tree
def drawTree(X, y, title):
    clf = tree.DecisionTreeClassifier()
    clf.fit(X, y)
    plt.figure(figsize=(15, 10))
    tree.plot_tree(clf)
    plt.title(title)
    plt.show()

    return 0

# make a label list of all 0, except the detected threat are 1
def labelList(num, threatIndices):
    y = [0 for x in range(num)]
    for i in threatIndices:
```

```
            y[i] =  1
        return y

    # convert object items in the list into strings
    def preprocess(df):

        # replace NAN with 0
        df = df.replace(np.nan, 0, regex=True)

        for column in df.columns:
            df[column] = list(map(str, df[column]))
        return df

    # fit the dataframe and record the label encoders for each column
    def labelEncoder_get(df):
        le_list = []
        for column in df.columns:
            le = preprocessing.LabelEncoder()
            le.fit(df[column])
            df[column] = le.transform(df[column])
            le_list.append(le)
        return (df, le_list)

    # input the list of label encoders and transform the dataframe
    def labelEncoder_trans(df, le_list):
        for i, column in enumerate(df.columns):
            le = le_list[i]
            df[column] = list(map(str, df[column]))
            df[column] = le.transform(df[column])
        return df

    # draw confusin matrix and print the classification report
    def draw_confusionMatrix(y_test, y_predict):
        confusion_matrix = metrics.confusion_matrix(y_test, y_predict)
        matrix_df = pd.DataFrame(confusion_matrix)
        ax = plt.axes()
        sns.set(font_scale=1.3)
        plt.figure(figsize=(10, 7))
        sns.heatmap(matrix_df, annot=True, fmt="g", ax=ax, cmap="magma")
        ax.set_title('Confusion Matrix - Decision Tree')
        ax.set_xlabel("Predicted Label", fontsize=15)
        ax.set_ylabel("Actual Label", fontsize=15)
        plt.show()
        print(metrics.classification_report(y_test, y_predict))

    # split the training and testing sets manually, make sure that positive data is
        in the testing set
    def train_test_split_m(X, y, positive_list, test_size=0.33):

        sample_len = int(len(X) * 0.33 - len(positive_list))

        X_test = X.iloc[positive_list]
        X = X.drop(positive_list)

        X_random = X.sample(sample_len)
        X = X.drop(X_random.index)
        X_test = pd.concat([X_test, X_random], axis=0, ignore_index=True)
        y_test = list([1 for i in range(len(positive_list))] + [0 for i in range(
            sample_len)])

        X_train = X
```

```
    y_train = [0 for i in range(len(X_train))]

    return (X_train, X_test, y_train, y_test)


## Evaluation

In[122]:


from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_score, StratifiedKFold

def evaluation(X, y):
    X = preprocess(X)
    X, le_dict = labelEncoder_get(X)
#     y = labelList(df.shape[0], [2754, 2756, 2842, 2843, 2844, 2845, 2846,
    2847])
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.33,
        random_state=0)

    # gini
    clf = DecisionTreeClassifier()
    clf.fit(X_train, y_train)
    y_predict = clf.predict(X_test)
    print("Balanced accuracy score: {:.2f}".format(metrics.
        balanced_accuracy_score(y_test, y_predict)))
    draw_confusionMatrix(y_test, y_predict)

    # entropy
    clf2 = DecisionTreeClassifier(criterion="entropy")
    clf2.fit(X_train, y_train)
    y_predict = clf2.predict(X_test)
    print("Balanced accuracy score: {:.2f}".format(metrics.
        balanced_accuracy_score(y_test, y_predict)))
    draw_confusionMatrix(y_test, y_predict)

    return 0


In[123]:


for i, x in enumerate(X):
    evaluation(x, Y[i])


### Confusion matrix
#
# reference:
# https://towardsdatascience.com/how-to-implement-and-evaluate-decision-tree-
    classifiers-from-scikit-learn-36ef7f037a78

## Decision Trees for each dataset

In[3]:


X = []
Y = []
```

```
X1 = df1[["Channel", "EventID", "ObjectName", "SubjectUserName", "TargetImage", "
    CallTrace", "ImageLoaded", "@timestamp", "ProcessGuid", "SourceProcessGUID"]]
X1 = preprocess_labelEncoder(X1)

Y1 = labelList(df1.shape[0], [2459, 2460, 2450, 2430, 2437, 2438, 2448])

drawTree(X1, Y1, "Decision Tree on the LSASS Memory Read Access")

X.append(X1)
Y.append(Y1)


# In[121]:


evaluation(X1, Y1)
evaluation_entropy(X1, Y1)


# In[4]:


# df2.describe()
X2 = df2[["Channel", "EventID", "StartModule", "StartFunction", "TargetFilename",
     "ImageLoaded", "Hostname"]]
X2 = preprocess_labelEncoder(X2)

Y2 = labelList(df2.shape[0], [496])

drawTree(X2, Y2, "Decision Tree on the DLL Process Injection Create Remote Thread
    ")

X.append(X2)
Y.append(Y2)


# In[32]:


evaluation(X2, Y2)


# In[5]:


X3 = df3[["Channel", "EventID", "AccessMask", "Properties", "SubjectUserName", "
    LogonType", "SubjectLogonId", "TargetLogonId"]]
X3 = preprocess_labelEncoder(X3)

Y3 = labelList(df3.shape[0], [5125, 5126, 5127, 5114])

drawTree(X3, Y3, "Decision Tree on the AD Object Access Replication")

X.append(X3)
Y.append(Y3)


# In[33]:
```

```
evaluation(X3, Y3)
```

 # In[6]:

```
X4 = df4[["Channel", "EventID", "ObjectServer", "AccessMask", "ObjectType", "
    AttributeLDAPDisplayName", "AttributeValue"]]
X4 = preprocess_labelEncoder(X4)

Y4 = labelList(df4.shape[0], [2754, 2756, 2842, 2843, 2844, 2845, 2846, 2847])

drawTree(X4, Y4, "Decision Tree on the AD Mod Directory Replication")

X.append(X4)
Y.append(Y4)
```

 # In[34]:

```
evaluation(X4, Y4)
```

 # In[7]:

```
df5.shape
```

 # In[8]:

```
df6.shape
X6 = df6[["Channel", "EventID", "NewProcessName", "ParentProcessName", "Image", "
    ParentImage", "ImageLoaded", "Description", "PipeName"]]
X6 = preprocess_labelEncoder(X6)

Y6 = labelList(df6.shape[0], [876, 1251, 1325, 1370, 1372, 327, 258, 1100, 1106,
    913])

drawTree(X6, Y6, "Decision Tree on the Local PowerShell Execution")

X.append(X6)
Y.append(Y6)
```

 # In[35]:

```
evaluation(X6, Y6)
```

 # In[9]:

```
df7.shape
```

 # In[10]:

```
df8.shape
X8 = df8[["Channel", "EventID", "Message", "DestPort", "LayerRTID", "
    ParentProcessName", "NewProcessName", "ParentImage", "Image", "DestinationPort
    ", "User"]]
X8 = preprocess_labelEncoder(X8)

Y8 = labelList(df8.shape[0], [678, 478, 745, 856, 535, 867, 689, 1415, 1418])

drawTree(X8, Y8, "Decision Tree on the Remote PowerShell Execution")


X.append(X8)
Y.append(Y8)


# In[37]:


evaluation(X8, Y8)


# In[11]:


df9.shape
X9 = df9[["Channel", "EventID", "Message", "Description", "ImageLoaded", "Image",
     "PipeName"]]
X9 = preprocess_labelEncoder(X9)

Y9 = labelList(df9.shape[0], [2, 4, 7, 8, 12, 768, 799])

drawTree(X9, Y9, "Decision Tree on the Alternate PowerShell Hosts")

X.append(X9)
Y.append(Y9)


# In[44]:


evaluation(X9, Y9)


# In[12]:


df10.shape
X10 = df10[["Channel", "EventID", "AccessMask", "ObjectName", "LogonType", "
    SubjectUserName", "Hostname", "ShareName", "RelativeTargetName", "
    SubjectLogonId", "TargetLogonId", "Message"]]
X10 = preprocess_labelEncoder(X10)

Y10 = labelList(df10.shape[0], [5279, 5283, 5287, 5291, 5269])

drawTree(X10, Y10, "Decision Tree on the Domain DPAPI Backup Key Extraction")

X.append(X10)
Y.append(Y10)
```

```
# In[45]:


evaluation(X10, Y10)


# In[13]:


df11.shape
X11 = df11[["Channel", "EventID", "ObjectType", "ObjectName"]]
X11 = preprocess_labelEncoder(X11)

Y11 = labelList(df11.shape[0], [5626, 5629, 5640, 5643, 5654, 5657, 5668, 5671])

drawTree(X11, Y11, "Decision Tree on the Reg Key Access Syskey")

X.append(X11)
Y.append(Y11)


# In[46]:


evaluation(X11, Y11)


# In[14]:


df12.shape
# X12 = df[["Channel", "EventID", "ObjectType", "ObjectName"]]
# X12 = preprocess_labelEncoder(X12)

# Y12 = labelList(12349, [])

# drawTree(X12, Y12, "Decision Tree on the )


# In[104]:


X13 = df13[["Channel", "EventID", "ParentProcessName", "TargetLogonId", "
    LogonType", "SubjectUserName", "ParentImage", "LogonId"]]
X13 = preprocess_labelEncoder(X13)

Y13 = labelList(df13.shape[0], [551, 368, 514])

drawTree(X13, Y13, "Decision Tree on the Remote WMI Execution")

X.append(X13)
Y.append(Y13)


# In[105]:


evaluation(X13, Y13)
```

 # In[16]:

```
X14 = df14[["Channel", "EventID"]]
X14 = preprocess_labelEncoder(X14)

Y14 = labelList(df14.shape[0], [447, 457, 592, 465, 5360, 5366])

drawTree(X14, Y14, "Decision Tree on the WMI Eventing" )

X.append(X14)
Y.append(Y14)
```

 # In[50]:

```
evaluation(X14, Y14)
```

 # In[17]:

```
X15 = df15[["Channel", "EventID", "ImageLoaded", "Image"]]
X15 = preprocess_labelEncoder(X15)

Y15 = labelList(df15.shape[0], [2168, 2172, 2194, 2232, 2235])

drawTree(X15, Y15, "Decision Tree on the WMI Module Load" )

X.append(X15)
Y.append(Y15)
```

 # In[51]:

```
evaluation(X15, Y15)
```

 # In[77]:

```
X16 = df16[["Channel", "EventID"]]
X16 = preprocess_labelEncoder(X16)

Y16 = labelList(df16.shape[0], [1212])

drawTree(X16, Y16, "Decision Tree on the Local Service Installation" )

X.append(X16)
Y.append(Y16)
```

 # In[79]:

```
evaluation(X16, Y16)
```

 # In[19]:

```
X17 = df17[["Channel", "EventID", "SubjectUserName", "SubjectLogonId", "
    TargetLogonId", "LogonType"]]
X17 = preprocess_labelEncoder(X17)

Y17 = labelList(df17.shape[0], [1212, 1028])

drawTree(X17, Y17, "Decision Tree on the Remote Service Creation" )

X.append(X17)
Y.append(Y17)
```

 # In[55]:

```
evaluation(X17, Y17)
```

 # In[20]:

```
X18 = df18[["Channel", "EventID","ObjectType", "ObjectName", "AccessMask", "
    SubjectLogonId", "PrivilegeList", "Application", "LayerRTID", "Image", "
    LogonType", "SubjectUserName", "TargetLogonId"]]
X18 = preprocess_labelEncoder(X18)

Y18 = labelList(df18.shape[0], [1071, 1374, 854, 1066, 1100, 1372, 1373, 1374,
    1375, 1368, 1070, 1068])

drawTree(X18, Y18, "Decision Tree on the Remote SCM Handle" )

X.append(X18)
Y.append(Y18)
```

 # In[56]:

```
evaluation(X18, Y18)
```

 # In[21]:

```
X19 = df19[["Channel", "EventID", "Image", "TargetFilename", "SourceImage", "
    TargetImage", "CallTrace", "ProcessGuid", "SourceProcessGUID", "LogonId"]]
X19 = preprocess_labelEncoder(X19)

Y19 = labelList(df19.shape[0], [4927, 4936, 3458])

drawTree(X19, Y19, "Decision Tree on the Remote Interactive Task Mgr LSASS Dump"
    )

X.append(X19)
Y.append(Y19)
```

 # In[57]:

```
evaluation(X19, Y19)
```

 # In[22]:

```
X20 = df20[["Channel", "EventID", "ObjectType", "ObjectName", "ProcessName", "
    SubjectLogonId", "ObjectValueName", "TargetObject"]]
X20 = preprocess_labelEncoder(X20)

Y20 = labelList(df20.shape[0], [550, 552, 553, 556, 581, 589, 592, 601, 642, 651,
     655, 665, 564, 579, 591, 611, 615, 376, 504, 505, 546, 549])

drawTree(X20, Y20, "Decision Tree on the Reg Mod Extended Net NTLM Downgrade" )

X.append(X20)
Y.append(Y20)
```

 # In[58]:

```
evaluation(X20, Y20)
```

 # In[23]:

```
X21 = df21[["Channel", "EventID", "ObjectType", "ObjectName", "ProcessName", "
    SubjectLogonId", "ObjectValueName", "TargetObject"]]
X21 = preprocess_labelEncoder(X21)

Y21 = labelList(df21.shape[0], [5117, 5118, 5202, 5203, 5204, 5240, 5241, 5242,
    5340, 5344])

drawTree(X21, Y21, "Decision Tree on the Access to Microphone Device" )

X.append(X21)
Y.append(Y21)
```

 # In[59]:

```
evaluation(X21, Y21)
```

 # In[24]:

```
X22 = df22[["Channel", "EventID", "Message", "Image", "NewProcessName", "
    ImageLoaded", "LogonType", "ProcessName", "ProcessGuid", "ProcessId"]]
X22 = preprocess_labelEncoder(X22)

Y22 = labelList(df22.shape[0], [157, 3307, 156, 1532, 1744, 1537, 1591, 1600,
    2151, 1799])

drawTree(X22, Y22, "Decision Tree on the Remote WMI Active Script Event Consumers
    " )
```

```
X.append(X22)
Y.append(Y22)
```

 # In[60]:

```
evaluation(X22, Y22)
```

 # In[25]:

```
X23 = df23[["Channel", "EventID", "RelativeTargetName", "AccessMask", "
    SubjectUserName", "Image", "TargetFilename", "ImageLoaded"]]
X23 = preprocess_labelEncoder(X23)

Y23 = labelList(df23.shape[0], [12889, 12872, 21264])

drawTree(X23, Y23, "Decision Tree on the Remote DCOM IErtUtil DLL Hijack" )

X.append(X23)
Y.append(Y23)
```

 # In[119]:

```
evaluation(X23, Y23)
```

 # In[26]:

```
X24 = df24[["Channel", "EventID", "RelativeTargetName", "SubjectUserName", "
    AccessMask", "Image", "TargetFilename", "ImageLoaded"]]
X24 = preprocess_labelEncoder(X24)

Y24 = labelList(df24.shape[0], [450, 445, 651])

drawTree(X24, Y24, "Decision Tree on the Remote WMI Wbemcomn DLL Hijack" )

X.append(X24)
Y.append(Y24)
```

 # In[63]:

```
evaluation(X24, Y24)
```

 # In[27]:

```
X25 = df25[["Channel", "EventID", "ShareName", "SubjectUserName", "SubjectLogonId
    ", "AccessMask", "Image", "RelativeTargetName", "TargetFilename"]]
X25 = preprocess_labelEncoder(X25)

Y25 = labelList(df25.shape[0], [58, 57, 61, 211])
```

```
drawTree(X25, Y25, "Decision Tree on the SMB Create Remot File" )

X.append(X25)
Y.append(Y25)


# In[64]:


evaluation(X25, Y25)


# In[28]:


X26 = df26[["Channel", "EventID", "Image", "CommandLine", "Signed", "SourceImage"
    , "ImageLoaded", "TargetFilename", "SourceProcessGuid", "ProcessGuid"]]
X26 = preprocess_labelEncoder(X26)

Y26 = labelList(df26.shape[0], [593, 612, 613, 87])

drawTree(X26, Y26, "Decision Tree on the Wuauclt Create Remote Threat Execution"
    )

X.append(X26)
Y.append(Y26)


# In[65]:


evaluation(X26, Y26)


#  ## Good Trees
#  2 datasest have good performance: "Reg Key Access Syskey" and "NTLM Downgrade"

# In[144]:


X11 = df11[["Channel", "EventID", "ObjectType", "ObjectName"]]
X11 = preprocess_labelEncoder(X11)

Y11 = labelList(df11.shape[0], [5626, 5629, 5640, 5643, 5654, 5657, 5668, 5671])

# evaluation
X = preprocess(X11)
X, le_dict = labelEncoder_get(X)
X_train, X_test, y_train, y_test = train_test_split(X, Y11, test_size = 0.33,
    random_state=0)

# gini
clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)
y_predict = clf.predict(X_test)
print("Balanced accuracy score: {:.2f}".format(metrics.balanced_accuracy_score(
    y_test, y_predict)))
draw_confusionMatrix(y_test, y_predict)


# In[149]:
```

```
# draw tree
plt.figure(figsize=(6, 5))
tree.plot_tree(clf, feature_names=X11.columns, fontsize=10)
plt.title("Decision Tree on the Reg Key Access Syskey")
plt.show()


# In[150]:


X20 = df20[["Channel", "EventID", "ObjectType", "ObjectName", "ProcessName", "
    SubjectLogonId", "ObjectValueName", "TargetObject"]]
X20 = preprocess_labelEncoder(X20)

Y20 = labelList(df20.shape[0], [550, 552, 553, 556, 581, 589, 592, 601, 642, 651,
     655, 665, 564, 579, 591, 611, 615, 376, 504, 505, 546, 549])

# evaluation
X = preprocess(X20)
X, le_dict = labelEncoder_get(X)
X_train, X_test, y_train, y_test = train_test_split(X, Y20, test_size = 0.33,
    random_state=0)

# gini
clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)
y_predict = clf.predict(X_test)
print("Balanced accuracy score: {:.2f}".format(metrics.balanced_accuracy_score(
    y_test, y_predict)))
draw_confusionMatrix(y_test, y_predict)


# In[155]:


# draw tree
plt.figure(figsize=(10, 8))
tree.plot_tree(clf, feature_names=X20.columns, fontsize=10)
plt.title("Decision Tree on the Reg Mod Extended Net NTLM Downgrade")
plt.show()


# In[139]:


from collections import Counter
Counter(X11['ObjectName']).items()


# In[ ]:


from sklearn import metrics
import seaborn as sns
confusion_matrix = metrics.confusion_matrix(y_test, y_predict)
matrix_df = pd.DataFrame(confusion_matrix)
ax = plt.axes()
sns.set(font_scale=1.3)
plt.figure(figsize=(10, 7))
```

```
sns.heatmap(matrix_df, annot=True, fmt="g", ax=ax, cmap="magma")
ax.set_title('Confusion Matrix - Decision Tree')
ax.set_xlabel("Predicted Label", fontsize=15)
# ax.set_xticklabels([''] + labels)
ax.set_ylabel("Actual Label", fontsize=15)
# ax.set_yticklabels(list(labels), rotation=0)
plt.show()
```

Listing 3: Scenario 2

```
#!/usr/bin/env python
# coding: utf-8

# # Decision Tree

# ## Datasets

# In[1]:


import os
import pandas as pd
import numpy as np
from pandas.io import json

df = []

datasetJSONPath = os.getcwd() + "\\empire_mimikatz_logonpasswords_2020
    -08-07103224.json"
df1 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

datasetJSONPath = os.getcwd() + "\\
    empire_dllinjection_LoadLibrary_CreateRemoteThread_2020-07-22000048.json"
df2 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

datasetJSONPath = os.getcwd() + "\\
    empire_dcsync_dcerpc_drsuapi_DsGetNCChanges_2020-09-21185829.json"
df3 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

datasetJSONPath = os.getcwd() + "\\
    empire_powerview_ldap_ntsecuritydescriptor_2020-09-21130944.json"
df4 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

datasetJSONPath = os.getcwd() + "\\empire_enable_rdp_2019-05-18203650.json"
df5 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

datasetJSONPath = os.getcwd() + "\\empire_launcher_vbs_2020-09-04160940.json"
df6 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

datasetJSONPath = os.getcwd() + "\\empire_wdigest_downgrade_2019-05-18201922.json
    "
df7 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

datasetJSONPath = os.getcwd() + "\\empire_psremoting_stager_2020-09-20170827.json
    "
df8 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

datasetJSONPath = os.getcwd() + "\\empire_psremoting_stager_2020-09-20170827.json
    "
df9 = json.read_json(path_or_buf=datasetJSONPath, lines=True)
```

```
datasetJSONPath = os.getcwd() + "\\
    empire_mimikatz_backupkeys_dcerpc_smb_lsarpc_2020-10-22143243.json"
df10 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

datasetJSONPath = os.getcwd() + "\\empire_mimikatz_sam_access_2020-09-22041117.
    json"
df11 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

datasetJSONPath = os.getcwd() + "\\empire_mimikatz_sam_access_2020-09-22041117.
    json"
df12 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

datasetJSONPath = os.getcwd() + "\\
    empire_wmi_dcerpc_wmi_IWbemServices_ExecMethod_2020-09-21001437.json"
df13 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

datasetJSONPath = os.getcwd() + "\\
    empire_wmi_local_event_subscriptions_elevated_user_2020-09-04164306.json"
df14 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

datasetJSONPath = os.getcwd() + "\\empire_psinject_PEinjection_2020-08-07143205.
    json"
df15 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

datasetJSONPath = os.getcwd() + "\\empire_psexec_dcerpc_tcp_svcctl_2020
    -09-20121608.json"
df16 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

datasetJSONPath = os.getcwd() + "\\empire_psexec_dcerpc_tcp_svcctl_2020
    -09-20121608.json"
df17 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

datasetJSONPath = os.getcwd() + "\\
    empire_find_localadmin_smb_svcctl_OpenSCManager_2020-09-22021559.json"
df18 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

datasetJSONPath = os.getcwd() + "\\rdp_interactive_taskmanager_lsass_dump_2020
    -09-22043748.json"
df19 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

datasetJSONPath = os.getcwd() + "\\empire_monologue_netntlm_downgrade_2019
    -12-25045202.json"
df20 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

datasetJSONPath = os.getcwd() + "\\msf_record_mic_2020-06-09225055.json"
df21 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

datasetJSONPath = os.getcwd() + "\\
    covenant_wmi_remote_event_subscription_ActiveScriptEventConsumers_2020
    -09-01214330.json"
df22 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

datasetJSONPath = os.getcwd() + "\\covenant_dcom_iertutil_dll_hijack_2020
    -10-09183000.json"
df23 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

datasetJSONPath = os.getcwd() + "\\covenant_wmi_wbemcomn_dll_hijack_2020
    -10-09173318.json"
df24 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

datasetJSONPath = os.getcwd() + "\\covenant_copy_smb_CreateRequest_2020
```

```
    -09-22145302.json"
df25 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

datasetJSONPath = os.getcwd() + "\\
    covenant_lolbin_wuauclt_createremotethread_2020-10-12183248.json"
df26 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

# 5, 7 have code error and 12 doesn't have any result for labelling
df = [df1, df2, df3, df4, df6, df8, df9, df10, df11, df13, df14, df15, df16, df17
    , df18, df19, df20, df21, df22, df23, df24, df25, df26]


# In[2]:


from sklearn import tree
from sklearn import preprocessing
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import metrics

# functions definition:

# convert object items in the list into strings
def preprocess_labelEncoder(df):

    le = preprocessing.LabelEncoder()

    # replace NAN with 0
    df = df.replace(np.nan, 0, regex=True)

    for column in df.columns:
        # if dtype is int, does not need to convert to string , e.g. EventID
        if isinstance(df[column][0], int) and df[column][0] != 0:
            le.fit(df[column])
            df[column] = le.transform(df[column])

        # convert to list of strings and use label encoder to label
        else:
            df[column] = list(map(str, df[column]))
            le.fit(df[column])
            df[column] = le.transform(df[column])


    return df

# draw the decision tree
def drawTree(X, y, title):
    clf = tree.DecisionTreeClassifier()
    clf.fit(X, y)
    plt.figure(figsize=(15, 10))
    tree.plot_tree(clf)
    plt.title(title)
    plt.show()

    return 0

# make a label list of all 0, except the detected threat are 1
def labelList(num, threatIndices):
    y = [0 for x in range(num)]
    for i in threatIndices:
```

```
        y[i] =  1
    return y

# convert object items in the list into strings
def preprocess(df):

    # replace NAN with 0
    df = df.replace(np.nan, 0, regex=True)

    for column in df.columns:
        df[column] = list(map(str, df[column]))
    return df

# fit the dataframe and record the label encoders for each column
def labelEncoder_get(df):
    le_list = []
    for column in df.columns:
        le = preprocessing.LabelEncoder()
        le.fit(df[column])
        df[column] = le.transform(df[column])
        le_list.append(le)
    return (df, le_list)

# input the list of label encoders and transform the dataframe
def labelEncoder_trans(df, le_list):
    for i, column in enumerate(df.columns):
        le = le_list[i]
        df[column] = list(map(str, df[column]))
        df[column] = le.transform(df[column])
    return df

# draw confusin matrix and print the classification report
def draw_confusionMatrix(y_test, y_predict):
    confusion_matrix = metrics.confusion_matrix(y_test, y_predict)
    matrix_df = pd.DataFrame(confusion_matrix)
    ax = plt.axes()
    sns.set(font_scale=1.3)
    plt.figure(figsize=(10, 7))
    sns.heatmap(matrix_df, annot=True, fmt="g", ax=ax, cmap="magma")
    ax.set_title('Confusion Matrix - Decision Tree')
    ax.set_xlabel("Predicted Label", fontsize=15)
    ax.set_ylabel("Actual Label", fontsize=15)
    plt.show()
    print(metrics.classification_report(y_test, y_predict))

# split the training and testing sets manually, make sure that positive data is
    in the testing set
def train_test_split_m(X, y, positive_list, test_size=0.33):

    sample_len = int(len(X) * 0.33 - len(positive_list))

    X_test = X.iloc[positive_list]
    X = X.drop(positive_list)

    X_random = X.sample(sample_len)
    X = X.drop(X_random.index)
    X_test = pd.concat([X_test, X_random], axis=0, ignore_index=True)
    y_test = list([1 for i in range(len(positive_list))] + [0 for i in range(
        sample_len)])

    X_train = X
```

```
    y_train = [0 for i in range(len(X_train))]

    return (X_train, X_test, y_train, y_test)


### Confusion matrix
#
# reference:
# https://towardsdatascience.com/how-to-implement-and-evaluate-decision-tree-
    classifiers-from-scikit-learn-36ef7f037a78

## Evaluation

# In[17]:


from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_score, StratifiedKFold

def evaluation(X, y):
    X = preprocess(X)
    X, le_dict = labelEncoder_get(X)
#     y = labelList(df.shape[0], [2754, 2756, 2842, 2843, 2844, 2845, 2846,
    2847])
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.33,
        random_state=0)

    # gini
    clf = DecisionTreeClassifier()
    clf.fit(X_train, y_train)
    y_predict = clf.predict(X_test)
    print("Balanced accuracy score: {:.2f}".format(metrics.
        balanced_accuracy_score(y_test, y_predict)))
    draw_confusionMatrix(y_test, y_predict)

#     # entropy
#     clf2 = DecisionTreeClassifier(criterion="entropy")
#     clf2.fit(X_train, y_train)
#     y_predict = clf2.predict(X_test)
#     print("Balanced accuracy score: {:.2f}".format(metrics.
    balanced_accuracy_score(y_test, y_predict)))
#     draw_confusionMatrix(y_test, y_predict)

    return 0


# In[4]:


# for i, x in enumerate(X):
#     evaluation(x, Y[i])


## Decision Trees for each dataset

# In[5]:


X = []
Y = []
```

```
X1 = df1[["Channel", "EventID", "ObjectName", "SubjectUserName", "TargetImage", "
    CallTrace", "ImageLoaded", "@timestamp", "ProcessGuid", "SourceProcessGUID"]]
X1 = preprocess_labelEncoder(X1)

Y1 = labelList(df1.shape[0], [2459, 2460, 2450, 2430, 2437, 2438, 2448])

# drawTree(X1, Y1, "Decision Tree on the LSASS Memory Read Access")

X.append(X1)
Y.append(Y1)


# In[11]:


oversample_ros(X1, Y1)


# In[14]:


# oversample_adasyn(X1, Y1)


# In[18]:


# df2.describe()
X2 = df2[["Channel", "EventID", "StartModule", "StartFunction", "TargetFilename",
     "ImageLoaded", "Hostname"]]
X2 = preprocess_labelEncoder(X2)

Y2 = labelList(df2.shape[0], [496])

# drawTree(X2, Y2, "Decision Tree on the DLL Process Injection Create Remote
    Thread")

X.append(X2)
Y.append(Y2)


# In[19]:


oversample_ros(X2, Y2)


# In[20]:


X3 = df3[["Channel", "EventID", "AccessMask", "Properties", "SubjectUserName", "
    LogonType", "SubjectLogonId", "TargetLogonId"]]
X3 = preprocess_labelEncoder(X3)

Y3 = labelList(df3.shape[0], [5125, 5126, 5127, 5114])

# drawTree(X3, Y3, "Decision Tree on the AD Object Access Replication")

X.append(X3)
Y.append(Y3)
```

```
# In[24]:


# oversample_adasyn(X3, Y3)


# In[23]:


X4 = df4[["Channel", "EventID", "ObjectServer", "AccessMask", "ObjectType", "
    AttributeLDAPDisplayName", "AttributeValue"]]
X4 = preprocess_labelEncoder(X4)

Y4 = labelList(df4.shape[0], [2754, 2756, 2842, 2843, 2844, 2845, 2846, 2847])

# drawTree(X4, Y4, "Decision Tree on the AD Mod Directory Replication")

X.append(X4)
Y.append(Y4)


# In[26]:


oversample_adasyn(X4, Y4)


# In[7]:


df5.shape


# In[27]:


df6.shape
X6 = df6[["Channel", "EventID", "NewProcessName", "ParentProcessName", "Image", "
    ParentImage", "ImageLoaded", "Description", "PipeName"]]
X6 = preprocess_labelEncoder(X6)

Y6 = labelList(df6.shape[0], [876, 1251, 1325, 1370, 1372, 327, 258, 1100, 1106,
    913])

# drawTree(X6, Y6, "Decision Tree on the Local PowerShell Execution")

X.append(X6)
Y.append(Y6)


# In[28]:


oversample(X6, Y6)


# In[9]:
```

```
df7.shape


# In[29]:


df8.shape
X8 = df8[["Channel", "EventID", "Message", "DestPort", "LayerRTID", "
    ParentProcessName", "NewProcessName", "ParentImage", "Image", "DestinationPort
    ", "User"]]
X8 = preprocess_labelEncoder(X8)

Y8 = labelList(df8.shape[0], [678, 478, 745, 856, 535, 867, 689, 1415, 1418])

# drawTree(X8, Y8, "Decision Tree on the Remote PowerShell Execution")


X.append(X8)
Y.append(Y8)


# In[30]:


oversample(X8, Y8)


# In[31]:


df9.shape
X9 = df9[["Channel", "EventID", "Message", "Description", "ImageLoaded", "Image",
     "PipeName"]]
X9 = preprocess_labelEncoder(X9)

Y9 = labelList(df9.shape[0], [2, 4, 7, 8, 12, 768, 799])

# drawTree(X9, Y9, "Decision Tree on the Alternate PowerShell Hosts")

X.append(X9)
Y.append(Y9)


# In[33]:


oversample_ros(X9, Y9)


# In[34]:


oversample_smote(X9, Y9)


# In[35]:


oversample_adasyn(X9, Y9)
```

```
# In[39]:


df10.shape
X10 = df10[["Channel", "EventID", "AccessMask", "ObjectName", "LogonType", "
    SubjectUserName", "Hostname", "ShareName", "RelativeTargetName", "
    SubjectLogonId", "TargetLogonId", "Message"]]
X10 = preprocess_labelEncoder(X10)

Y10 = labelList(df10.shape[0], [5279, 5283, 5287, 5291, 5269])

# drawTree(X10, Y10, "Decision Tree on the Domain DPAPI Backup Key Extraction")

X.append(X10)
Y.append(Y10)


# In[45]:


oversample_ros(X10, Y10)


# In[44]:


# oversample_smote(X10, Y10)


# In[46]:


# df11.shape
# X11 = df11[["Channel", "EventID", "ObjectType", "ObjectName"]]
# X11 = preprocess_labelEncoder(X11)

# Y11 = labelList(df11.shape[0], [5626, 5629, 5640, 5643, 5654, 5657, 5668,
    5671])

# # drawTree(X11, Y11, "Decision Tree on the Reg Key Access Syskey")

# X.append(X11)
# Y.append(Y11)


# In[47]:


# evaluation(X11, Y11)


# In[14]:


df12.shape
# X12 = df[["Channel", "EventID", "ObjectType", "ObjectName"]]
# X12 = preprocess_labelEncoder(X12)

# Y12 = labelList(12349, [])

# drawTree(X12, Y12, "Decision Tree on the )
```

```
# In[48]:


X13 = df13[["Channel", "EventID", "ParentProcessName", "TargetLogonId", "
    LogonType", "SubjectUserName", "ParentImage", "LogonId"]]
X13 = preprocess_labelEncoder(X13)

Y13 = labelList(df13.shape[0], [551, 368, 514])

# drawTree(X13, Y13, "Decision Tree on the Remote WMI Execution")

X.append(X13)
Y.append(Y13)


# In[49]:


oversample(X13, Y13)


# In[50]:


X14 = df14[["Channel", "EventID"]]
X14 = preprocess_labelEncoder(X14)

Y14 = labelList(df14.shape[0], [447, 457, 592, 465, 5360, 5366])

# drawTree(X14, Y14, "Decision Tree on the WMI Eventing" )

X.append(X14)
Y.append(Y14)


# In[53]:


oversample_ros(X14, Y14)


# In[56]:


# oversample_adasyn(X14, Y14)


# In[57]:


X15 = df15[["Channel", "EventID", "ImageLoaded", "Image"]]
X15 = preprocess_labelEncoder(X15)

Y15 = labelList(df15.shape[0], [2168, 2172, 2194, 2232, 2235])

# drawTree(X15, Y15, "Decision Tree on the WMI Module Load" )

X.append(X15)
Y.append(Y15)
```

```
# In[59]:


oversample_ros(X15, Y15)


# In[62]:


# oversample_adasyn(X15, Y15)


# In[63]:


X16 = df16[["Channel", "EventID"]]
X16 = preprocess_labelEncoder(X16)

Y16 = labelList(df16.shape[0], [1212])

# drawTree(X16, Y16, "Decision Tree on the Local Service Installation" )

X.append(X16)
Y.append(Y16)


# In[67]:


oversample_ros(X16, Y16)


# In[68]:


X17 = df17[["Channel", "EventID", "SubjectUserName", "SubjectLogonId", "
    TargetLogonId", "LogonType"]]
X17 = preprocess_labelEncoder(X17)

Y17 = labelList(df17.shape[0], [1212, 1028])

# drawTree(X17, Y17, "Decision Tree on the Remote Service Creation" )

X.append(X17)
Y.append(Y17)


# In[70]:


oversample_ros(X17, Y17)


# In[72]:


# oversample_smote(X17, Y17)
```

```
# In[73]:


X18 = df18[["Channel", "EventID","ObjectType", "ObjectName", "AccessMask", "
    SubjectLogonId", "PrivilegeList", "Application", "LayerRTID", "Image", "
    LogonType", "SubjectUserName", "TargetLogonId"]]
X18 = preprocess_labelEncoder(X18)

Y18 = labelList(df18.shape[0], [1071, 1374, 854, 1066, 1100, 1372, 1373, 1374,
    1375, 1368, 1070, 1068])

# drawTree(X18, Y18, "Decision Tree on the Remote SCM Handle" )

X.append(X18)
Y.append(Y18)


# In[74]:


oversample(X18, Y18)


# In[75]:


X19 = df19[["Channel", "EventID", "Image", "TargetFilename", "SourceImage", "
    TargetImage", "CallTrace", "ProcessGuid", "SourceProcessGUID", "LogonId"]]
X19 = preprocess_labelEncoder(X19)

Y19 = labelList(df19.shape[0], [4927, 4936, 3458])

# drawTree(X19, Y19, "Decision Tree on the Remote Interactive Task Mgr LSASS Dump
    " )

X.append(X19)
Y.append(Y19)


# In[77]:


oversample_ros(X19, Y19)


# In[79]:


# oversample_smote(X19,Y19)


# In[80]:


# X20 = df20[["Channel", "EventID", "ObjectType", "ObjectName", "ProcessName", "
    SubjectLogonId", "ObjectValueName", "TargetObject"]]
# X20 = preprocess_labelEncoder(X20)

# Y20 = labelList(df20.shape[0], [550, 552, 553, 556, 581, 589, 592, 601, 642,
    651, 655, 665, 564, 579, 591, 611, 615, 376, 504, 505, 546, 549])
```

```
# drawTree(X20, Y20, "Decision Tree on the Reg Mod Extended Net NTLM Downgrade" )

# X.append(X20)
# Y.append(Y20)


# In[81]:


# evaluation(X20, Y20)


# In[82]:


X21 = df21[["Channel", "EventID", "ObjectType", "ObjectName", "ProcessName", "
    SubjectLogonId", "ObjectValueName", "TargetObject"]]
X21 = preprocess_labelEncoder(X21)

Y21 = labelList(df21.shape[0], [5117, 5118, 5202, 5203, 5204, 5240, 5241, 5242,
    5340, 5344])

# drawTree(X21, Y21, "Decision Tree on the Access to Microphone Device" )

X.append(X21)
Y.append(Y21)


# In[83]:


oversample(X21, Y21)


# In[84]:


X22 = df22[["Channel", "EventID", "Message", "Image", "NewProcessName", "
    ImageLoaded", "LogonType", "ProcessName", "ProcessGuid", "ProcessId"]]
X22 = preprocess_labelEncoder(X22)

Y22 = labelList(df22.shape[0], [157, 3307, 156, 1532, 1744, 1537, 1591, 1600,
    2151, 1799])

# drawTree(X22, Y22, "Decision Tree on the Remote WMI Active Script Event
    Consumers" )

X.append(X22)
Y.append(Y22)


# In[85]:


oversample(X22, Y22)


# In[86]:


X23 = df23[["Channel", "EventID", "RelativeTargetName", "AccessMask", "
```

```
    SubjectUserName", "Image", "TargetFilename", "ImageLoaded"]]
X23 = preprocess_labelEncoder(X23)

Y23 = labelList(df23.shape[0], [12889, 12872, 21264])

# drawTree(X23, Y23, "Decision Tree on the Remote DCOM IErtUtil DLL Hijack" )

X.append(X23)
Y.append(Y23)


# In[89]:


oversample_ros(X23, Y23)


# In[91]:


# oversample_smote(X23, Y23)


# In[92]:


X24 = df24[["Channel", "EventID", "RelativeTargetName", "SubjectUserName", "
    AccessMask", "Image", "TargetFilename", "ImageLoaded"]]
X24 = preprocess_labelEncoder(X24)

Y24 = labelList(df24.shape[0], [450, 445, 651])

# drawTree(X24, Y24, "Decision Tree on the Remote WMI Wbemcomn DLL Hijack" )

X.append(X24)
Y.append(Y24)


# In[94]:


oversample_ros(X24, Y24)


# In[96]:


# oversample_smote(X24, Y24)


# In[97]:


X25 = df25[["Channel", "EventID", "ShareName", "SubjectUserName", "SubjectLogonId
    ", "AccessMask", "Image", "RelativeTargetName", "TargetFilename"]]
X25 = preprocess_labelEncoder(X25)

Y25 = labelList(df25.shape[0], [58, 57, 61, 211])

# drawTree(X25, Y25, "Decision Tree on the SMB Create Remot File" )
```

```
X.append(X25)
Y.append(Y25)


# In[99]:


oversample_ros(X25, Y25)


# In[101]:


# oversample_smote(X25, Y25)


# In[102]:


X26 = df26[["Channel", "EventID", "Image", "CommandLine", "Signed", "SourceImage"
    , "ImageLoaded", "TargetFilename", "SourceProcessGuid", "ProcessGuid"]]
X26 = preprocess_labelEncoder(X26)

Y26 = labelList(df26.shape[0], [593, 612, 613, 87])

# drawTree(X26, Y26, "Decision Tree on the Wuauclt Create Remote Threat Execution
    " )

X.append(X26)
Y.append(Y26)


# In[104]:


oversample_ros(X26, Y26)


# In[106]:


# oversample_smote(X26,Y26)


# ### Data Imbalance - Oversampling
# (1) RandomOverSampler
#
# (2) SMOTE (Synthetic Minority Oversampling Technique)
#
# (3) ADASYN

# In[3]:


from imblearn.over_sampling import RandomOverSampler
from imblearn.over_sampling import SMOTE
from imblearn.over_sampling import ADASYN

def oversample(X, y):
    X = preprocess(X)
    X, le_dict = labelEncoder_get(X)
```

```
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.33,
        random_state=0)

    # gini
    clf = DecisionTreeClassifier()
#     clf.fit(X_train, y_train)
#     y_predict = clf.predict(X_test)
#     print("Balanced accuracy score: {:.2f}".format(metrics.
    balanced_accuracy_score(y_test, y_predict)))
#     draw_confusionMatrix(y_test, y_predict)

    # ros
    ros = RandomOverSampler(random_state=42)
    X_resampled, y_resampled = ros.fit_resample(X_train, y_train)
    clf.fit(X_resampled, y_resampled)
    y_predict = clf.predict(X_test)
    print("ROS - Balanced accuracy score: {:.2f}".format(metrics.
        balanced_accuracy_score(y_test, y_predict)))
    draw_confusionMatrix(y_test, y_predict)

    # smote
    smote = SMOTE()
    X_resampled, y_resampled = smote.fit_resample(X_train, y_train)
    clf.fit(X_resampled, y_resampled)
    y_predict = clf.predict(X_test)
    print("SMOTE - Balanced accuracy score: {:.2f}".format(metrics.
        balanced_accuracy_score(y_test, y_predict)))
    draw_confusionMatrix(y_test, y_predict)

    # adasyn
    adasyn = ADASYN()
    X_resampled, y_resampled = adasyn.fit_resample(X_train, y_train)
    clf.fit(X_resampled, y_resampled)
    y_predict = clf.predict(X_test)
    print("ADASYN - Balanced accuracy score: {:.2f}".format(metrics.
        balanced_accuracy_score(y_test, y_predict)))
    draw_confusionMatrix(y_test, y_predict)

    return 0


# In[19]:


# ROS (first split then resample)
from imblearn.over_sampling import RandomOverSampler

def oversample_ros(X, y):
    ros = RandomOverSampler(random_state=42)
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.33,
        random_state=42)
    X_resampled, y_resampled = ros.fit_resample(X_train, y_train)

    clf_ros = DecisionTreeClassifier()
    clf_ros.fit(X_resampled, y_resampled)
    y_predict = clf_ros.predict(X_test)
    print("ROS - Balanced accuracy score: {:.2f}".format(metrics.
        balanced_accuracy_score(y_test, y_predict)))
    draw_confusionMatrix(y_test, y_predict)
#     draw_confusion(y_test, y_predict, "Confusion Matrix - ROS")
```

```
# # cross-validation
# balanced_accuracies = cv_resampled(ros, clf_ros, X, y)
# print(''Average Balanced Accuracy:'', balanced_accuracies.mean())
    return 0


# In[20]:


# SMOTE (first split then resample)
from imblearn.over_sampling import SMOTE

def oversample_smote(X, y):
    smote = SMOTE()
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.33,
        random_state=42)
    X_resampled, y_resampled = smote.fit_resample(X_train, y_train)

    clf_smote = DecisionTreeClassifier()
    clf_smote.fit(X_resampled, y_resampled)
    y_predict = clf_smote.predict(X_test)
    print("SMOTE - Balanced accuracy score: {:.2f}".format(metrics.
        balanced_accuracy_score(y_test, y_predict)))
    draw_confusionMatrix(y_test, y_predict)
#     draw_confusion(y_test, y_predict, "Confusion Matrix - SMOTE")


#     # cross-validation
#     balanced_accuracies = cv_resampled(smote, clf_smote, X, y)
#     print(''Average Balanced Accuracy:'', balanced_accuracies.mean())
    return 0


# In[21]:


# ADASYN (first split then resample)
from imblearn.over_sampling import ADASYN

def oversample_adasyn(X, y):

    adasyn = ADASYN()
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.33,
        random_state=42)
    X_resampled, y_resampled = adasyn.fit_resample(X_train, y_train)

    clf_adasyn = DecisionTreeClassifier()
    clf_adasyn.fit(X_resampled, y_resampled)
    y_predict = clf_adasyn.predict(X_test)
    print("ADASYN - Balanced accuracy score: {:.2f}".format(metrics.
        balanced_accuracy_score(y_test, y_predict)))
    draw_confusionMatrix(y_test, y_predict)
#     draw_confusion(y_test, y_predict, "Confusion Matrix - ADASYN")


#     # cross-validation
#     balanced_accuracies = cv_resampled(adasyn, clf_adasyn, X, y)
#     print(''Average Balanced Accuracy:'', balanced_accuracies.mean())
    return 0
```

```
## Others

In[149]:


# draw tree
plt.figure(figsize=(6, 5))
tree.plot_tree(clf, feature_names=X11.columns, fontsize=10)
plt.title("Decision Tree on the Reg Key Access Syskey")
plt.show()


In[ ]:


# Naive random oversampling
from sklearn.datasets import make_classification
from imblearn.over_sampling import RandomOverSampler
from collections import Counter

data_label = label['label'].tolist()
ros = RandomOverSampler(random_state=0)
X_resampled, y_resampled = ros.fit_resample(data, data_label)

print("y before oversampling: ", sorted(Counter(data_label).items()))
print("y after oversampling: ", sorted(Counter(y_resampled).items()))

# Tree
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled,
    test_size = 0.33, random_state=42)


clf_ros = tree.DecisionTreeClassifier(min_samples_split=5) # max_depth=45
clf_ros.fit(X_train, y_train)
y_predict = clf_ros.predict(X_test)

plt.figure(figsize=(200, 80))
tree.plot_tree(clf_ros, feature_names=totalFeatures, fontsize=10)
plt.title("Classification Decision Tree with ROS data")
plt.savefig('ROSTree.png', dpi=100, transparent=True)


In[ ]:


confusion_matrix = metrics.confusion_matrix(y_test, y_predict)
matrix_df = pd.DataFrame(confusion_matrix)
ax = plt.axes()
sns.set(font_scale=1.3)
plt.figure(figsize=(10, 7))
sns.heatmap(matrix_df, annot=True, fmt="g", ax=ax, cmap="magma")
ax.set_title('Confusion Matrix - Decision Tree with ROS data')
ax.set_xlabel("Predicted Label", fontsize=15)
ax.set_ylabel("Actual Label", fontsize=15)
plt.show()
print(metrics.classification_report(y_test, y_predict))


In[ ]:
```

```
# SMOTE
from imblearn.over_sampling import SMOTE
X_resampled, y_resampled = SMOTE().fit_resample(data, data_label)
print("y after oversampling: ", sorted(Counter(y_resampled).items()))

# Tree
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled,
    test_size = 0.33, random_state=42)

clf_smote = tree.DecisionTreeClassifier(min_samples_split=5) # max_depth=45
clf_smote.fit(X_train, y_train)
y_predict = clf_smote.predict(X_test)

plt.figure(figsize=(200, 80))
tree.plot_tree(clf_smote, feature_names=totalFeatures, fontsize=10)
plt.title("Classification Decision Tree with SMOTE data")
plt.savefig('SMOTETree.png', dpi=100, transparent=True)


# In[ ]:


confusion_matrix = metrics.confusion_matrix(y_test, y_predict)
matrix_df = pd.DataFrame(confusion_matrix)
ax = plt.axes()
sns.set(font_scale=1.3)
plt.figure(figsize=(10, 7))
sns.heatmap(matrix_df, annot=True, fmt="g", ax=ax, cmap="magma")
ax.set_title('Confusion Matrix - Decision Tree with SMOTE data')
ax.set_xlabel("Predicted Label", fontsize=15)
ax.set_ylabel("Actual Label", fontsize=15)
plt.show()
print(metrics.classification_report(y_test, y_predict))


# In[ ]:


# ADASYN
from imblearn.over_sampling import ADASYN
X_resampled, y_resampled = ADASYN().fit_resample(data, data_label)
print("y after oversamoling: ", sorted(Counter(y_resampled).items()))

# Tree
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled,
    test_size = 0.33, random_state=42)

clf_adasyn = tree.DecisionTreeClassifier(min_samples_split=5) # max_depth=45
clf_adasyn.fit(X_train, y_train)
y_predict = clf_adasyn.predict(X_test)

plt.figure(figsize=(200, 80))
tree.plot_tree(clf_adasyn, feature_names=totalFeatures, fontsize=10)
plt.title("Classification Decision Tree with ADASYN data")
plt.savefig('ADASYNTree.png', dpi=100, transparent=True)


# In[ ]:
```

```
confusion_matrix = metrics.confusion_matrix(y_test, y_predict)
matrix_df = pd.DataFrame(confusion_matrix)
ax = plt.axes()
sns.set(font_scale=1.3)
plt.figure(figsize=(10, 7))
sns.heatmap(matrix_df, annot=True, fmt="g", ax=ax, cmap="magma")
ax.set_title('Confusion Matrix - Decision Tree with ADASYN data')
ax.set_xlabel("Predicted Label", fontsize=15)
ax.set_ylabel("Actual Label", fontsize=15)
plt.show()
print(metrics.classification_report(y_test, y_predict))
```

Listing 4: Scenario 3

```
#!/usr/bin/env python
# coding: utf-8

## Decision Tree

### Datasets

# In[1]:


import os
import pandas as pd
import numpy as np
from pandas.io import json

df = []

datasetJSONPath = os.getcwd() + "\\empire_mimikatz_logonpasswords_2020
    -08-07103224.json"
df1 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

datasetJSONPath = os.getcwd() + "\\
    empire_dllinjection_LoadLibrary_CreateRemoteThread_2020-07-22000048.json"
df2 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

datasetJSONPath = os.getcwd() + "\\
    empire_dcsync_dcerpc_drsuapi_DsGetNCChanges_2020-09-21185829.json"
df3 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

datasetJSONPath = os.getcwd() + "\\
    empire_powerview_ldap_ntsecuritydescriptor_2020-09-21130944.json"
df4 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

datasetJSONPath = os.getcwd() + "\\empire_enable_rdp_2019-05-18203650.json"
df5 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

datasetJSONPath = os.getcwd() + "\\empire_launcher_vbs_2020-09-04160940.json"
df6 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

datasetJSONPath = os.getcwd() + "\\empire_wdigest_downgrade_2019-05-18201922.json
    "
df7 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

datasetJSONPath = os.getcwd() + "\\empire_psremoting_stager_2020-09-20170827.json
    "
df8 = json.read_json(path_or_buf=datasetJSONPath, lines=True)
```

```
datasetJSONPath = os.getcwd() + "\\empire_psremoting_stager_2020-09-20170827.json
    "
df9 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

datasetJSONPath = os.getcwd() + "\\
    empire_mimikatz_backupkeys_dcerpc_smb_lsarpc_2020-10-22143243.json"
df10 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

datasetJSONPath = os.getcwd() + "\\empire_mimikatz_sam_access_2020-09-22041117.
    json"
df11 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

datasetJSONPath = os.getcwd() + "\\empire_mimikatz_sam_access_2020-09-22041117.
    json"
df12 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

datasetJSONPath = os.getcwd() + "\\
    empire_wmi_dcerpc_wmi_IWbemServices_ExecMethod_2020-09-21001437.json"
df13 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

datasetJSONPath = os.getcwd() + "\\
    empire_wmi_local_event_subscriptions_elevated_user_2020-09-04164306.json"
df14 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

datasetJSONPath = os.getcwd() + "\\empire_psinject_PEinjection_2020-08-07143205.
    json"
df15 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

datasetJSONPath = os.getcwd() + "\\empire_psexec_dcerpc_tcp_svcctl_2020
    -09-20121608.json"
df16 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

datasetJSONPath = os.getcwd() + "\\empire_psexec_dcerpc_tcp_svcctl_2020
    -09-20121608.json"
df17 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

datasetJSONPath = os.getcwd() + "\\
    empire_find_localadmin_smb_svcctl_OpenSCManager_2020-09-22021559.json"
df18 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

datasetJSONPath = os.getcwd() + "\\rdp_interactive_taskmanager_lsass_dump_2020
    -09-22043748.json"
df19 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

datasetJSONPath = os.getcwd() + "\\empire_monologue_netntlm_downgrade_2019
    -12-25045202.json"
df20 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

datasetJSONPath = os.getcwd() + "\\msf_record_mic_2020-06-09225055.json"
df21 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

datasetJSONPath = os.getcwd() + "\\
    covenant_wmi_remote_event_subscription_ActiveScriptEventConsumers_2020
    -09-01214330.json"
df22 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

datasetJSONPath = os.getcwd() + "\\covenant_dcom_iertutil_dll_hijack_2020
    -10-09183000.json"
df23 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

datasetJSONPath = os.getcwd() + "\\covenant_wmi_wbemcomn_dll_hijack_2020
```

```
    -10-09173318.json"
df24 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

datasetJSONPath = os.getcwd() + "\\covenant_copy_smb_CreateRequest_2020
    -09-22145302.json"
df25 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

datasetJSONPath = os.getcwd() + "\\
    covenant_lolbin_wuauclt_createremotethread_2020-10-12183248.json"
df26 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

# 5, 7 have code error and 12 doesn't have any result for labelling
df = [df1, df2, df3, df4, df6, df8, df9, df10, df11, df13, df14, df15, df16, df17
    , df18, df19, df20, df21, df22, df23, df24, df25, df26]


# In[2]:


from sklearn import tree
from sklearn import preprocessing
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import metrics
from sklearn.model_selection import train_test_split

# functions definition:

# convert object items in the list into strings
def preprocess_labelEncoder(df):

    le = preprocessing.LabelEncoder()

    # replace NAN with 0
    df = df.replace(np.nan, 0, regex=True)

    for column in df.columns:
        # if dtype is int, does not need to convert to string , e.g. EventID
        if isinstance(df[column][0], int) and df[column][0] != 0:
            le.fit(df[column])
            df[column] = le.transform(df[column])

        # convert to list of strings and use label encoder to label
        else:
            df[column] = list(map(str, df[column]))
            le.fit(df[column])
            df[column] = le.transform(df[column])


    return df

# draw the decision tree
def drawTree(X, y, title):
    clf = tree.DecisionTreeClassifier()
    clf.fit(X, y)
    plt.figure(figsize=(15, 10))
    tree.plot_tree(clf)
    plt.title(title)
    plt.show()

    return 0
```

```
# make a label list of all 0, except the detected threat are 1
def labelList(num, threatIndices):
    y = [0 for x in range(num)]
    for i in threatIndices:
        y[i] =  1
    return y

# convert object items in the list into strings
def preprocess(df):

    # replace NAN with 0
    df = df.replace(np.nan, 0, regex=True)

    for column in df.columns:
        df[column] = list(map(str, df[column]))
    return df

# fit the dataframe and record the label encoders for each column
def labelEncoder_get(df):
    le_list = []
    for column in df.columns:
        le = preprocessing.LabelEncoder()
        le.fit(df[column])
        df[column] = le.transform(df[column])
        le_list.append(le)
    return (df, le_list)

# input the list of label encoders and transform the dataframe
def labelEncoder_trans(df, le_list):
    for i, column in enumerate(df.columns):
        le = le_list[i]
        df[column] = list(map(str, df[column]))
        df[column] = le.transform(df[column])
    return df

# draw confusin matrix and print the classification report
def draw_confusionMatrix(y_test, y_predict):
    confusion_matrix = metrics.confusion_matrix(y_test, y_predict)
    matrix_df = pd.DataFrame(confusion_matrix)
    ax = plt.axes()
    sns.set(font_scale=1.3)
    plt.figure(figsize=(10, 7))
    sns.heatmap(matrix_df, annot=True, fmt="g", ax=ax, cmap="magma")
    ax.set_title('Confusion Matrix - Decision Tree')
    ax.set_xlabel("Predicted Label", fontsize=15)
    ax.set_ylabel("Actual Label", fontsize=15)
    plt.show()
    print(metrics.classification_report(y_test, y_predict))

# split the training and testing sets manually, make sure that positive data is
    in the testing set
def train_test_split_m(X, y, positive_list, test_size=0.33):

    sample_len = int(len(X) * 0.33 - len(positive_list))

    X_test = X.iloc[positive_list]
    X = X.drop(positive_list)

    X_random = X.sample(sample_len)
    X = X.drop(X_random.index)
```

```
    X_test = pd.concat([X_test, X_random], axis=0, ignore_index=True)
    y_test = list([1 for i in range(len(positive_list))] + [0 for i in range(
        sample_len)])

    X_train = X
    y_train = [0 for i in range(len(X_train))]

    return (X_train, X_test, y_train, y_test)


### Confusion matrix
#
# reference:
# https://towardsdatascience.com/how-to-implement-and-evaluate-decision-tree-
    classifiers-from-scikit-learn-36ef7f037a78

## Evaluation

# In[3]:


from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_score, StratifiedKFold

def evaluation(X, y):
    X = preprocess(X)
    X, le_dict = labelEncoder_get(X)
#     y = labelList(df.shape[0], [2754, 2756, 2842, 2843, 2844, 2845, 2846,
    2847])
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.33,
        random_state=0)

    # gini
    clf = DecisionTreeClassifier()
    clf.fit(X_train, y_train)
    y_predict = clf.predict(X_test)
    print("Balanced accuracy score: {:.2f}".format(metrics.
        balanced_accuracy_score(y_test, y_predict)))
    draw_confusionMatrix(y_test, y_predict)


    return 0


## DNN Model

# In[61]:


import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras.initializers import GlorotUniform
from tensorflow.keras.optimizers import Adam
from sklearn.preprocessing import StandardScaler

from imblearn.over_sampling import RandomOverSampler
from imblearn.over_sampling import SMOTE
from imblearn.over_sampling import ADASYN
from sklearn import metrics
```

```
def DNN_model(input_len, layer=4):

    # Define the input shape
    input_shape = (input_len,)

    # Define the number of neurons in each hidden layer
    if layer == 1:
        hidden_counts = [input_len*3]
    elif layer == 2:
        hidden_counts = [input_len*3, int(input_len*3/2)]
    elif layer == 3:
        hidden_counts = [input_len*3, int(input_len*3/2), int(input_len*3/2/2)]
    elif layer == 4:
        hidden_counts = [input_len*3, int(input_len*3/2), int(input_len*3/2)/2,
            int(input_len*3/2/2/2)]
    else:
        hidden_counts = [input_len*3, int(input_len*3/2), int(input_len*3/2/2),
            int(input_len*3/2/2/2), int(input_len*3/2/2/2/2)]

    # Define the number of output neurons
    num_classes = 1

    # Define the activation function for the hidden layers
    activation_fn = 'relu'

    # Define the weight initializer
    initializer = GlorotUniform(seed=42)

    # Define the optimizer
    optimizer = Adam()

    # Define the model architecture
    model = tf.keras.Sequential()

    # Add the input layer
    model.add(layers.InputLayer(input_shape=input_shape))

    # Add the hidden layers
    for i, count in enumerate(hidden_counts):
        model.add(layers.Dense(count, activation=activation_fn,
            kernel_initializer=initializer, name=f'hidden_{i+1}'))

    # Add the output layer
    # model.add(layers.Dense(num_classes, activation='softmax',
        kernel_initializer=initializer, name='output'))
    model.add(layers.Dense(num_classes, activation='sigmoid', kernel_initializer=
        initializer, name='output'))


    # Compile the model
    # model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics
        =['accuracy'])
    model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['
        accuracy', 'AUC'])

    return model


def DNN_classifier(X, y, resample=2, hidden_layers=4):

    # preprocess
```

```
    X = preprocess(X)
    X, le_dict = labelEncoder_get(X)

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.33,
        random_state=42)

    # oversampling
    if resample == 1: # ros
        ros = RandomOverSampler(random_state=42)
        X_resampled, y_resampled = ros.fit_resample(X_train, y_train)

    elif resample == 2: # smote
        smote = SMOTE()
        X_resampled, y_resampled = smote.fit_resample(X_train, y_train)
    else: # adasyn
        adasyn = ADASYN()
        X_resampled, y_resampled = adasyn.fit_resample(X_train, y_train)

    # z-score scaling
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X_resampled)

    ## DNN Model
    model = DNN_model(len(X.columns))
    hist = model.fit(X_scaled, y_resampled, verbose=2, epochs=50)
    plt.plot(hist.history['accuracy'])
    plt.show()

    # prediction
    y_predict = model.predict(X_test)
    y_predict_class = np.where(y_predict >= 0.5, 1, 0) # use 0.5 as threshold
    draw_confusionMatrix(y_test, y_predict_class)

    return 0


# In[ ]:


# prediction
from sklearn import metrics

y_predict = model.predict(X_test)
y_predict_class = np.where(y_predict >= 0.5, 1, 0) # use 0.5 as threshold
draw_confusionMatrix(y_test, y_predict_class)


# ## DNN for each dataset

# In[5]:


X = []
Y = []

X1 = df1[["Channel", "EventID", "ObjectName", "SubjectUserName", "TargetImage", "
    CallTrace", "ImageLoaded", "@timestamp", "ProcessGuid", "SourceProcessGUID"]]
X1 = preprocess_labelEncoder(X1)

Y1 = labelList(df1.shape[0], [2459, 2460, 2450, 2430, 2437, 2438, 2448])
```

```
# drawTree(X1, Y1, "Decision Tree on the LSASS Memory Read Access")

X.append(X1)
Y.append(Y1)


# In[16]:


y = np.array(Y1)

# 4 hidden layers
DNN_classifier(X1, y)


# In[14]:


# oversample_adasyn(X1, Y1)


# In[18]:


# df2.describe()
X2 = df2[["Channel", "EventID", "StartModule", "StartFunction", "TargetFilename",
     "ImageLoaded", "Hostname"]]
X2 = preprocess_labelEncoder(X2)

Y2 = labelList(df2.shape[0], [496])

# drawTree(X2, Y2, "Decision Tree on the DLL Process Injection Create Remote
    Thread")

X.append(X2)
Y.append(Y2)


# In[31]:


# oversample_ros(X2, Y2)


# In[20]:


X3 = df3[["Channel", "EventID", "AccessMask", "Properties", "SubjectUserName", "
    LogonType", "SubjectLogonId", "TargetLogonId"]]
X3 = preprocess_labelEncoder(X3)

Y3 = labelList(df3.shape[0], [5125, 5126, 5127, 5114])

# drawTree(X3, Y3, "Decision Tree on the AD Object Access Replication")

X.append(X3)
Y.append(Y3)


# In[24]:
```

```
# oversample_adasyn(X3, Y3)


# In[23]:


X4 = df4[["Channel", "EventID", "ObjectServer", "AccessMask", "ObjectType", "
    AttributeLDAPDisplayName", "AttributeValue"]]
X4 = preprocess_labelEncoder(X4)

Y4 = labelList(df4.shape[0], [2754, 2756, 2842, 2843, 2844, 2845, 2846, 2847])

# drawTree(X4, Y4, "Decision Tree on the AD Mod Directory Replication")

X.append(X4)
Y.append(Y4)


# In[17]:


# oversample_adasyn(X4, Y4)


# In[7]:


df5.shape


# In[22]:


df6.shape
X6 = df6[["Channel", "EventID", "NewProcessName", "ParentProcessName", "Image", "
    ParentImage", "ImageLoaded", "Description", "PipeName"]]
X6 = preprocess_labelEncoder(X6)

Y6 = labelList(df6.shape[0], [876, 1251, 1325, 1370, 1372, 327, 258, 1100, 1106,
    913])

# drawTree(X6, Y6, "Decision Tree on the Local PowerShell Execution")

X.append(X6)
Y.append(Y6)


# In[24]:


y = np.array(Y6)
DNN_classifier(X6, y, hidden_layers=1)


# In[25]:


DNN_classifier(X6, y, hidden_layers=2)
```

```
# In[26]:


DNN_classifier(X6, y, hidden_layers=3)


# In[27]:


DNN_classifier(X6, y, hidden_layers=4)


# In[28]:


DNN_classifier(X6, y, hidden_layers=5)


# In[9]:


df7.shape


# In[29]:


df8.shape
X8 = df8[["Channel", "EventID", "Message", "DestPort", "LayerRTID", "
    ParentProcessName", "NewProcessName", "ParentImage", "Image", "DestinationPort
    ", "User"]]
X8 = preprocess_labelEncoder(X8)

Y8 = labelList(df8.shape[0], [678, 478, 745, 856, 535, 867, 689, 1415, 1418])

# drawTree(X8, Y8, "Decision Tree on the Remote PowerShell Execution")


X.append(X8)
Y.append(Y8)


# In[33]:


y = np.array(Y8)
DNN_classifier(X8, y, hidden_layers=1)


# In[34]:


DNN_classifier(X8, y, hidden_layers=2)


# In[35]:


DNN_classifier(X8, y, hidden_layers=3)
```

```
# In[36]:


DNN_classifier(X8, y, hidden_layers=4)


# In[37]:


DNN_classifier(X8, y, hidden_layers=5)


# In[62]:


df9.shape
X9 = df9[["Channel", "EventID", "Message", "Description", "ImageLoaded", "Image",
    "PipeName"]]
X9 = preprocess_labelEncoder(X9)

Y9 = labelList(df9.shape[0], [2, 4, 7, 8, 12, 768, 799])

# drawTree(X9, Y9, "Decision Tree on the Alternate PowerShell Hosts")

X.append(X9)
Y.append(Y9)


# In[63]:


y = np.array(Y9)
DNN_classifier(X9, y, hidden_layers=1)


# In[64]:


DNN_classifier(X9, y, hidden_layers=2)


# In[65]:


DNN_classifier(X9, y, hidden_layers=3)


# In[66]:


DNN_classifier(X9, y, hidden_layers=4)


# In[67]:


DNN_classifier(X9, y, hidden_layers=5)


# In[39]:
```

```
df10.shape
X10 = df10[["Channel", "EventID", "AccessMask", "ObjectName", "LogonType", "
    SubjectUserName", "Hostname", "ShareName", "RelativeTargetName", "
    SubjectLogonId", "TargetLogonId", "Message"]]
X10 = preprocess_labelEncoder(X10)

Y10 = labelList(df10.shape[0], [5279, 5283, 5287, 5291, 5269])

# drawTree(X10, Y10, "Decision Tree on the Domain DPAPI Backup Key Extraction")

X.append(X10)
Y.append(Y10)


# In[45]:


oversample_ros(X10, Y10)


# In[44]:


# oversample_smote(X10, Y10)


# In[46]:


# df11.shape
# X11 = df11[["Channel", "EventID", "ObjectType", "ObjectName"]]
# X11 = preprocess_labelEncoder(X11)

# Y11 = labelList(df11.shape[0], [5626, 5629, 5640, 5643, 5654, 5657, 5668,
    5671])

# # drawTree(X11, Y11, "Decision Tree on the Reg Key Access Syskey")

# X.append(X11)
# Y.append(Y11)


# In[47]:


# evaluation(X11, Y11)


# In[14]:


df12.shape
# X12 = df[["Channel", "EventID", "ObjectType", "ObjectName"]]
# X12 = preprocess_labelEncoder(X12)

# Y12 = labelList(12349, [])

# drawTree(X12, Y12, "Decision Tree on the )
```

```
# In[48]:


X13 = df13[["Channel", "EventID", "ParentProcessName", "TargetLogonId", "
    LogonType", "SubjectUserName", "ParentImage", "LogonId"]]
X13 = preprocess_labelEncoder(X13)

Y13 = labelList(df13.shape[0], [551, 368, 514])

# drawTree(X13, Y13, "Decision Tree on the Remote WMI Execution")

X.append(X13)
Y.append(Y13)


# In[49]:


oversample(X13, Y13)


# In[50]:


X14 = df14[["Channel", "EventID"]]
X14 = preprocess_labelEncoder(X14)

Y14 = labelList(df14.shape[0], [447, 457, 592, 465, 5360, 5366])

# drawTree(X14, Y14, "Decision Tree on the WMI Eventing" )

X.append(X14)
Y.append(Y14)


# In[53]:


oversample_ros(X14, Y14)


# In[56]:


# oversample_adasyn(X14, Y14)


# In[57]:


X15 = df15[["Channel", "EventID", "ImageLoaded", "Image"]]
X15 = preprocess_labelEncoder(X15)

Y15 = labelList(df15.shape[0], [2168, 2172, 2194, 2232, 2235])

# drawTree(X15, Y15, "Decision Tree on the WMI Module Load" )

X.append(X15)
Y.append(Y15)
```

```
# In[59]:


oversample_ros(X15, Y15)


# In[62]:


# oversample_adasyn(X15, Y15)


# In[63]:


X16 = df16[["Channel", "EventID"]]
X16 = preprocess_labelEncoder(X16)

Y16 = labelList(df16.shape[0], [1212])

# drawTree(X16, Y16, "Decision Tree on the Local Service Installation" )

X.append(X16)
Y.append(Y16)


# In[67]:


oversample_ros(X16, Y16)


# In[68]:


X17 = df17[["Channel", "EventID", "SubjectUserName", "SubjectLogonId", "
    TargetLogonId", "LogonType"]]
X17 = preprocess_labelEncoder(X17)

Y17 = labelList(df17.shape[0], [1212, 1028])

# drawTree(X17, Y17, "Decision Tree on the Remote Service Creation" )

X.append(X17)
Y.append(Y17)


# In[70]:


oversample_ros(X17, Y17)


# In[72]:


# oversample_smote(X17, Y17)


# In[41]:
```

```
X18 = df18[["Channel", "EventID","ObjectType", "ObjectName", "AccessMask", "
    SubjectLogonId", "PrivilegeList", "Application", "LayerRTID", "Image", "
    LogonType", "SubjectUserName", "TargetLogonId"]]
X18 = preprocess_labelEncoder(X18)

Y18 = labelList(df18.shape[0], [1071, 1374, 854, 1066, 1100, 1372, 1373, 1374,
    1375, 1368, 1070, 1068])

# drawTree(X18, Y18, "Decision Tree on the Remote SCM Handle" )

X.append(X18)
Y.append(Y18)


# In[42]:


y = np.array(Y18)
DNN_classifier(X18, y, hidden_layers=1)


# In[43]:


DNN_classifier(X18, y, hidden_layers=2)


# In[44]:


DNN_classifier(X18, y, hidden_layers=3)


# In[45]:


DNN_classifier(X18, y, hidden_layers=4)


# In[46]:


DNN_classifier(X18, y, hidden_layers=5)


# In[75]:


X19 = df19[["Channel", "EventID", "Image", "TargetFilename", "SourceImage", "
    TargetImage", "CallTrace", "ProcessGuid", "SourceProcessGUID", "LogonId"]]
X19 = preprocess_labelEncoder(X19)

Y19 = labelList(df19.shape[0], [4927, 4936, 3458])

# drawTree(X19, Y19, "Decision Tree on the Remote Interactive Task Mgr LSASS Dump
    " )

X.append(X19)
Y.append(Y19)
```

```
# In[77]:


oversample_ros(X19, Y19)


# In[79]:


# oversample_smote(X19, Y19)


# In[80]:


# X20 = df20[["Channel", "EventID", "ObjectType", "ObjectName", "ProcessName", "
    SubjectLogonId", "ObjectValueName", "TargetObject"]]
# X20 = preprocess_labelEncoder(X20)

# Y20 = labelList(df20.shape[0], [550, 552, 553, 556, 581, 589, 592, 601, 642,
    651, 655, 665, 564, 579, 591, 611, 615, 376, 504, 505, 546, 549])

# drawTree(X20, Y20, "Decision Tree on the Reg Mod Extended Net NTLM Downgrade" )

# X.append(X20)
# Y.append(Y20)


# In[81]:


# evaluation(X20, Y20)


# In[47]:


X21 = df21[["Channel", "EventID", "ObjectType", "ObjectName", "ProcessName", "
    SubjectLogonId", "ObjectValueName", "TargetObject"]]
X21 = preprocess_labelEncoder(X21)

Y21 = labelList(df21.shape[0], [5117, 5118, 5202, 5203, 5204, 5240, 5241, 5242,
    5340, 5344])

# drawTree(X21, Y21, "Decision Tree on the Access to Microphone Device" )

X.append(X21)
Y.append(Y21)


# In[48]:


y = np.array(Y21)
DNN_classifier(X21, y, hidden_layers=1)


# In[50]:
```

```
DNN_classifier(X21, y, hidden_layers=2)


# In[51]:


DNN_classifier(X21, y, hidden_layers=3)


# In[52]:


DNN_classifier(X21, y, hidden_layers=4)


# In[53]:


DNN_classifier(X21, y, hidden_layers=5)


# In[54]:


X22 = df22[["Channel", "EventID", "Message", "Image", "NewProcessName", "
    ImageLoaded", "LogonType", "ProcessName", "ProcessGuid", "ProcessId"]]
X22 = preprocess_labelEncoder(X22)

Y22 = labelList(df22.shape[0], [157, 3307, 156, 1532, 1744, 1537, 1591, 1600,
    2151, 1799])

# drawTree(X22, Y22, "Decision Tree on the Remote WMI Active Script Event
    Consumers" )

X.append(X22)
Y.append(Y22)


# In[56]:


y = np.array(Y22)
DNN_classifier(X22, y, hidden_layers=1)


# In[57]:


DNN_classifier(X22, y, hidden_layers=2)


# In[58]:


DNN_classifier(X22, y, hidden_layers=3)


# In[59]:


DNN_classifier(X22, y, hidden_layers=4)
```

```
# In[60]:


DNN_classifier(X22, y, hidden_layers=5)


# In[86]:


X23 = df23[["Channel", "EventID", "RelativeTargetName", "AccessMask", "
    SubjectUserName", "Image", "TargetFilename", "ImageLoaded"]]
X23 = preprocess_labelEncoder(X23)

Y23 = labelList(df23.shape[0], [12889, 12872, 21264])

# drawTree(X23, Y23, "Decision Tree on the Remote DCOM IErtUtil DLL Hijack" )

X.append(X23)
Y.append(Y23)


# In[89]:


oversample_ros(X23, Y23)


# In[91]:


# oversample_smote(X23, Y23)


# In[92]:


X24 = df24[["Channel", "EventID", "RelativeTargetName", "SubjectUserName", "
    AccessMask", "Image", "TargetFilename", "ImageLoaded"]]
X24 = preprocess_labelEncoder(X24)

Y24 = labelList(df24.shape[0], [450, 445, 651])

# drawTree(X24, Y24, "Decision Tree on the Remote WMI Wbemcomn DLL Hijack" )

X.append(X24)
Y.append(Y24)


# In[94]:


oversample_ros(X24, Y24)


# In[96]:


# oversample_smote(X24, Y24)
```

```
# In[97]:


X25 = df25[["Channel", "EventID", "ShareName", "SubjectUserName", "SubjectLogonId
    ", "AccessMask", "Image", "RelativeTargetName", "TargetFilename"]]
X25 = preprocess_labelEncoder(X25)

Y25 = labelList(df25.shape[0], [58, 57, 61, 211])

# drawTree(X25, Y25, "Decision Tree on the SMB Create Remot File" )

X.append(X25)
Y.append(Y25)


# In[99]:


oversample_ros(X25, Y25)


# In[101]:


# oversample_smote(X25, Y25)


# In[102]:


X26 = df26[["Channel", "EventID", "Image", "CommandLine", "Signed", "SourceImage"
    , "ImageLoaded", "TargetFilename", "SourceProcessGuid", "ProcessGuid"]]
X26 = preprocess_labelEncoder(X26)

Y26 = labelList(df26.shape[0], [593, 612, 613, 87])

# drawTree(X26, Y26, "Decision Tree on the Wuauclt Create Remote Threat Execution
    " )

X.append(X26)
Y.append(Y26)


# In[104]:


oversample_ros(X26, Y26)


# In[106]:


# oversample_smote(X26,Y26)
```

Listing 5: Scenario 4 - Example

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:
```

```
import os
import pandas as pd
import numpy as np
import tensorflow as tf
from pandas.io import json
from tensorflow import keras
from tensorflow.keras import layers
from sklearn import preprocessing

datasetJSONPath = os.getcwd() + "/datasets/empire_psremoting_stager_2020
    -09-20170827.json"
df = json.read_json(path_or_buf=datasetJSONPath, lines=True)

df.shape


# In[13]:


# functions definition

from collections import Counter
import seaborn as sns
from sklearn import metrics
import matplotlib.pyplot as plt

# Convert integers to strings
def int_to_str(df):
    for column in df.columns:
        # if dtype is int, convert to string , e.g. EventID
        if isinstance(df[column][0], (int, np.int32, np.int64)):
            df[column] = list(map(str, df[column]))

    return df

# Count NaN in each column in the dataframe
def count_nan(df):
    for column in df.columns:
        num = df[column].isna().sum()
        print(column, str(num))
    return 0

# make a label list of all 0, except the detected threat are 1
def labelList(num, threatIndices):
    y = [0 for x in range(num)]
    for i in threatIndices:
        y[i] =  1
    return y

# Count the items in data frame
def count_items(df):
    for column in df.columns:
        item = df[column]
        print(column, ": ", sorted(Counter(item).items()))

    return 0

# replicate specified data to the end of the dataset
def pd_replicate(X, y, rep_list, times=1):
```

```
    for index in rep_list:
        for i in range(times):

            X = pd.concat([X, pd.DataFrame(X.iloc[index]).T], axis=0,
                ignore_index=True)

            y.append(1)

    return (X, y)

# convert object items in the list into strings
def preprocess(df):

    # replace NAN with 0
    df = df.replace(np.nan, 0, regex=True)

    for column in df.columns:
        df[column] = list(map(str, df[column]))
    return df

# fit the dataframe and record the label encoders for each column
def labelEncoder_get(df):
    le_dict = {}
    for column in df.columns:
        le = preprocessing.LabelEncoder()
        le.fit(df[column])
        df[column] = le.transform(df[column])
        le_dict[column] = le
    return (df, le_dict)

# input the list of label encoders and transform the dataframe
def labelEncoder_trans(df, le_dict):
    for i, column in enumerate(df.columns):
        le = le_dict[column]
        df[column] = list(map(str, df[column]))
        df[column] = le.transform(df[column])
    return df

# draw confusin matrix and print the classification report
def draw_confusionMatrix(y_test, y_predict):
    confusion_matrix = metrics.confusion_matrix(y_test, y_predict)
    matrix_df = pd.DataFrame(confusion_matrix)
    ax = plt.axes()
    sns.set(font_scale=1.3)
    plt.figure(figsize=(10, 7))
    sns.heatmap(matrix_df, annot=True, fmt="g", ax=ax, cmap="magma")
    ax.set_title('Confusion Matrix - Decision Tree')
    ax.set_xlabel("Predicted Label", fontsize=15)
    ax.set_ylabel("Actual Label", fontsize=15)
    plt.show()
    print(metrics.classification_report(y_test, y_predict))


# Randomly pick data from results and combine them
def random_generator(data_list, pick_nums):

    generated_data = pd.DataFrame()

    for i, data in enumerate(data_list):
        num = pick_nums[i]
        data_selected = data.sample(num)
```

```
        generated_data = pd.concat([generated_data, data_selected], axis=0,
            ignore_index=True)

    return generated_data

# split the training and testing sets manually, make sure that positive data is
    in the testing set
def train_test_split_m(X, y, positive_list, test_size=0.33):

    sample_len = int(len(X) * 0.33 - len(positive_list))

    X_test = X.iloc[positive_list]
    X = X.drop(positive_list)

    X_random = X.sample(sample_len)
    X = X.drop(X_random.index)
    X_test = pd.concat([X_test, X_random], axis=0, ignore_index=True)
    y_test = list([1 for i in range(len(positive_list))] + [0 for i in range(
        sample_len)])

    X_train = X
    y_train = [0 for i in range(len(X_train))]

    return (X_train, X_test, y_train, y_test)


# In[3]:


features = ["Channel", "EventID", "Message", "Description", "ImageLoaded", "Image
    ", "PipeName"]
df = df[features]

# change integers to strings to get a better performance in generating new data
df = int_to_str(df)
df.shape


# ## Replicate positive data
#
# There are more negative data than positive data, and results from some
    analytics (e.g. analytics with merging) are less possible to be generated. So
    we replicate specific positive data to make CTGAN bias to them.

# ## Use CTGAN to generate new data

# In[12]:


# from sdv.tabular import CTGAN

# model = CTGAN(cuda=True)
# model.fit(df)
# save and reuse the model
# model.save('PwshAlternateHosts.pkl')

# model = CTGAN.load(os.getcwd() + '/models/PwshAlternateHosts.pkl')


# In[4]:
```

```
from sdv.metadata import SingleTableMetadata
from sdv.single_table import CTGANSynthesizer

metadata = SingleTableMetadata()
metadata.detect_from_dataframe(data=df)

synthesizer = CTGANSynthesizer(metadata,cuda=True)

synthesizer.fit(df)

# save and reuse the model
synthesizer.save(filepath=os.getcwd() + '/models/PwshAlternateHosts.pkl')


# In[5]:


new_data = synthesizer.sample(num_rows=1000000)
# new_data = model.sample(num_rows=1000000)
new_data.head()


## Use hunter program on new data

# In[6]:


# Analytic I
I = (
new_data[features]

[((new_data['Channel'] == 'Microsoft-Windows-PowerShell/Operational')|(new_data['
    Channel'] == 'Windows PowerShell')) #
    & ((new_data['EventID'] == '400')|(df['EventID'] == '4103'))
    & (~new_data['Message'].str.contains('.*Host Application%powershell.*', regex
       =True, na=False))
]
)
I


# In[9]:


len(I)


# In[7]:


# Analytic II
II  = (
new_data[features]

[(new_data['Channel'] == 'Microsoft-Windows-Sysmon/Operational')
    & (new_data['EventID'] == '7')
    & (
      (new_data['Description'].str.lower() == 'system.management.automation')
      | (new_data['ImageLoaded'].str.contains('.*system.management.automation.*',
          regex=True))
```

```
    )
    & (~new_data['Image'].str.lower().str.endswith('powershell.exe', na=False))
]
)
II


# In[8]:


# Analytic III
III =(
new_data[features]

[(new_data['Channel'] == 'Microsoft-Windows-Sysmon/Operational')
    & (new_data['EventID'] == '17')
    & (new_data['PipeName'].str.lower().str.startswith('\pshost', na=False))
    & (~new_data['Image'].str.lower().str.endswith('powershell.exe', na=False))
]
)
III


### Generate data
# Randomly pick data from each analytic's results, and combine them as the
    generated dataset.

# In[10]:


# randomly pick from results of analytics
results = [I, II, III]
pick_nums = [37, 311, 1500] # 2737 * 2/3 = 1824
gen_data = random_generator(results, pick_nums)

gen_data.shape


# In[ ]:


gen_data.head()


## Decision Tree

### Preprocess data

# In[28]:


# Preprocessing
from sklearn.preprocessing import MinMaxScaler

# scaler = MinMaxScaler()
# df = scaler.fit_transform(df)


### Original data

# In[14]:
```

```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_score, StratifiedKFold

X = preprocess(df)
X, le_dict = labelEncoder_get(X)
y = labelList(df.shape[0], [2, 4, 7, 8, 12, 768, 799])

clf = DecisionTreeClassifier()
# X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.33,
    random_state=42)
X_train, X_test, y_train, y_test = train_test_split_m(X, y, [2, 4, 7, 8, 12, 768,
    799])

clf.fit(X_train, y_train)
y_predict = clf.predict(X_test)
print("Balanced accuracy score: {:.2f}".format(metrics.balanced_accuracy_score(
    y_test, y_predict)))
draw_confusionMatrix(y_test, y_predict)


### Original data with generated data
Generated data should only be added into trianing set.

In[16]:


X_gen = preprocess(gen_data)
X_gen = labelEncoder_trans(X_gen, le_dict)
add generated data to the original training set
X_train_new = pd.concat([X_train, X_gen], axis=0)


generated data are all positive
for i in range(len(gen_data)):
    y_train.append(1)

clf.fit(X_train_new, y_train)
y_predict = clf.predict(X_test)
print("Balanced accuracy score: {:.2f}".format(metrics.balanced_accuracy_score(
    y_test, y_predict)))
draw_confusionMatrix(y_test, y_predict)


In[19]:


from sklearn import tree
draw the tree
plt.figure(figsize=(30, 15))
tree.plot_tree(clf, feature_names=features, fontsize=10)
plt.title("Decision Tree - PwshAlternateHosts")
plt.savefig(r'DecisionTree/PwshAlternateHosts.png', dpi=100, transparent=True)


## Save generated data

In[21]:
```

```
# # extract to JSON file
# dataset.reset_index(drop=True, inplace=True)
gen_data.to_json(r'generated_datasets/empire_psremoting_stager_new2.json', orient
    ='records')

# dataset = json.read_json(path_or_buf=os.getcwd()+ "\\generated_datasets"+ "\\
    empire_dllinjection_LoadLibrary_CreateRemoteThread_new.json")
# dataset.head()
```

Listing 6: Scenario 5

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


import os
import pandas as pd
import numpy as np
import tensorflow as tf
from pandas.io import json
from tensorflow import keras
from tensorflow.keras import layers
from sklearn import preprocessing

from collections import Counter
import seaborn as sns
from sklearn import metrics
import matplotlib.pyplot as plt


from tensorflow.keras.initializers import GlorotUniform
from tensorflow.keras.optimizers import Adam
from sklearn.preprocessing import StandardScaler

from sklearn import metrics


# In[2]:


# functions definition

# Convert integers to strings
def int_to_str(df):
    for column in df.columns:
        # if dtype is int, convert to string , e.g. EventID
        if isinstance(df[column][0], (int, np.int32, np.int64)):
            df[column] = list(map(str, df[column]))

    return df

# Count NaN in each column in the dataframe
def count_nan(df):
    for column in df.columns:
        num = df[column].isna().sum()
        print(column, str(num))
    return 0
```

```
# make a label list of all 0, except the detected threat are 1
def labelList(num, threatIndices):
    y = [0 for x in range(num)]
    for i in threatIndices:
        y[i] =  1
    return y

# Count the items in data frame
def count_items(df):
    for column in df.columns:
        item = df[column]
        print(column, ": ", sorted(Counter(item).items()))

    return 0

# replicate specified data to the end of the dataset
def pd_replicate(X, y, rep_list, times=1):
    for index in rep_list:
        for i in range(times):

            X = pd.concat([X, pd.DataFrame(X.iloc[index]).T], axis=0,
                ignore_index=True)

            y.append(1)

    return (X, y)

# convert object items in the list into strings
def preprocess(df):

    # replace NAN with 0
    df = df.replace(np.nan, 0, regex=True)

    for column in df.columns:
        df[column] = list(map(str, df[column]))
    return df

# fit the dataframe and record the label encoders for each column
def labelEncoder_get(df):
    le_list = []
    for column in df.columns:
        le = preprocessing.LabelEncoder()
        le.fit(df[column])
        df[column] = le.transform(df[column])
        le_list.append(le)
    return (df, le_list)

# input the list of label encoders and transform the dataframe
def labelEncoder_trans(df, le_list):
    for i, column in enumerate(df.columns):
        le = le_list[i]
        df[column] = list(map(str, df[column]))
        df[column] = le.transform(df[column])
    return df


# draw confusin matrix and print the classification report
def draw_confusionMatrix(y_test, y_predict):
    confusion_matrix = metrics.confusion_matrix(y_test, y_predict)
    matrix_df = pd.DataFrame(confusion_matrix)
    ax = plt.axes()
```

```
    sns.set(font_scale=1.3)
    plt.figure(figsize=(10, 7))
    sns.heatmap(matrix_df, annot=True, fmt="g", ax=ax, cmap="magma")
    ax.set_title('Confusion Matrix - DNN')
    ax.set_xlabel("Predicted Label", fontsize=15)
    ax.set_ylabel("Actual Label", fontsize=15)
    plt.show()
    print(metrics.classification_report(y_test, y_predict))


# Randomly pick data from results and combine them
def random_generator(data_list, pick_nums):

    generated_data = pd.DataFrame()

    for i, data in enumerate(data_list):
        num = pick_nums[i]
        data_selected = data.sample(num)
        generated_data = pd.concat([generated_data, data_selected], axis=0,
            ignore_index=True)

    return generated_data

# split the training and testing sets manually, make sure that positive data is
    in the testing set
def train_test_split_m(X, y, positive_list, test_size=0.33):

    sample_len = int(len(X) * 0.33 - len(positive_list))

    X_test = X.iloc[positive_list]
    X = X.drop(positive_list)

    X_random = X.sample(sample_len)
    X = X.drop(X_random.index)
    X_test = pd.concat([X_test, X_random], axis=0, ignore_index=True)
    y_test = list([1 for i in range(len(positive_list))] + [0 for i in range(
        sample_len)])

    X_train = X
    y_train = [0 for i in range(len(X_train))]

    return (X_train, X_test, y_train, y_test)


# ## DNN Model
#
# The neural networks that were investigated by this research
# had hidden layer counts between one and five. Neuron counts
# for the first hidden layer were three times the number of input
# neurons needed for the feature vector. The neuron counts for
# subsequent hidden layers were half of the preceding layer.
# Using this methodology a 50 input neural network would
# contain hidden counts of [150, 75, 37, 18] for a four hidden
# layer network.
#
# All hidden layers made use of the rectified linear unit
# (ReLU) transfer function. The adaptive moment
# estimation (Adam) was used to train the neural network.
# Weight initialization was accomplished using the Xavier
# algorithm. The feature vector was optimized using a
# feature ranking algorithm that was developed specifically for
```

```
# TensorFlow.
#
# Refernces: *Deep Learning for Prioritizing and Responding to Intrusion
    Detection Alerts* https://ieeexplore.ieee.org/abstract/document/8170757
#
# *Batch Normalization: Accelerating Deep Network Training by Reducing Internal
    Covariate Shift* http://proceedings.mlr.press/v37/ioffe15.html
#

# In[44]:


def DNN_model(input_len, layer=4):

    # Define the input shape
    input_shape = (input_len,)

    # Define the number of neurons in each hidden layer
    if layer == 1:
        hidden_counts = [input_len*3]
    elif layer == 2:
        hidden_counts = [input_len*3, int(input_len*3/2)]
    elif layer == 3:
        hidden_counts = [input_len*3, int(input_len*3/2), int(input_len*3/2/2)]
    elif layer == 4:
        hidden_counts = [input_len*3, int(input_len*3/2), int(input_len*3/2)/2,
            int(input_len*3/2/2/2)]
    else:
        hidden_counts = [input_len*3, int(input_len*3/2), int(input_len*3/2/2),
            int(input_len*3/2/2/2), int(input_len*3/2/2/2/2)]

    # Define the number of output neurons
    num_classes = 1

    # Define the activation function for the hidden layers
    activation_fn = 'relu'

    # Define the weight initializer
    initializer = GlorotUniform(seed=42)

    # Define the optimizer
    optimizer = Adam()

    # Define the model architecture
    model = tf.keras.Sequential()

    # Add the input layer
    model.add(layers.InputLayer(input_shape=input_shape))

    # Add the hidden layers
    for i, count in enumerate(hidden_counts):
        model.add(layers.Dense(count, activation=activation_fn,
            kernel_initializer=initializer, name=f'hidden_{i+1}'))

    # Add the output layer
    # model.add(layers.Dense(num_classes, activation='softmax',
        kernel_initializer=initializer, name='output'))
    model.add(layers.Dense(num_classes, activation='sigmoid', kernel_initializer=
        initializer, name='output'))
```

```
    # Compile the model
    # model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics
        =['accuracy'])
    model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['
        accuracy'])#, 'AUC'

    return model


def DNN_classifier(df, y, y_index, gen_data, hidden_layers=3, manually=False):

    # preprocess
    X = preprocess(df)
    X, le_dict = labelEncoder_get(X)
    if manually:
        X_train, X_test, y_train, y_test = train_test_split_m(X, y, y_index)
    else:
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
            0.33, random_state=42)


    # add generated data to the original training set
    X_gen = preprocess(gen_data)
    X_gen = labelEncoder_trans(X_gen, le_dict)
    X_train_new = pd.concat([X_train, X_gen], axis=0)
    # generated data are all positive
    for i in range(len(gen_data)):
        y_train.append(1)

    # z-score scaling
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X_train_new)

    ## DNN Model
    model = DNN_model(len(X.columns))
    y_train = np.array(y_train) #
    hist = model.fit(X_scaled, y_train, verbose=2, epochs=50)
    plt.plot(hist.history['accuracy'])
    plt.show()

    # prediction
    y_predict = model.predict(X_test)
    y_predict_class = np.where(y_predict >= 0.5, 1, 0) # use 0.5 as threshold
    draw_confusionMatrix(y_test, y_predict_class)

    return 0


# In[83]:


# AD Object
# read the original dataset
datasetJSONPath = os.getcwd() + "\\datasets\\
    empire_dcsync_dcerpc_drsuapi_DsGetNCChanges_2020-09-21185829.json"
df1 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

features = ["Channel", "EventID", "AccessMask", "Properties", "SubjectUserName",
    "LogonType", "SubjectLogonId", "TargetLogonId"]
df1 = df1[features]
```

```
Y1 = labelList(df1.shape[0], [5125, 5126, 5127, 5114])

# read the generated data
gen_data1 = json.read_json(path_or_buf=os.getcwd()+ r'\generated_datasets\
    empire_dcsync_dcerpc_drsuapi_DsGetNCChanges_new.json')

print(df1.shape, gen_data1.shape)


# In[84]:


DNN_classifier(df1, Y1, [5125, 5126, 5127, 5114], gen_data1, hidden_layers=5,
    manually=True)


# In[85]:


# Local PS
datasetJSONPath = os.getcwd() + "\\datasets\\empire_launcher_vbs_2020
    -09-04160940.json"
df2 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

features = ["Channel", "EventID", "NewProcessName", "ParentProcessName", "Image",
     "ImageLoaded", "Description", "PipeName"] # "ParentImage",
df2 = df2[features]

Y2 = labelList(df2.shape[0], [876, 1251, 1325, 1370, 1372, 327, 258, 1100, 1106,
    913])

gen_data2 = json.read_json(os.getcwd() + "\\generated_datasets\\
    empire_launcher_vbs_new.json")

DNN_classifier(df2, Y2, [876, 1251, 1325, 1370, 1372, 327, 258, 1100, 1106, 913],
     gen_data2, hidden_layers=5, manually=True)


# In[86]:


# PS Remote
datasetJSONPath = os.getcwd() + "\\datasets\\empire_psremoting_stager_2020
    -09-20170827.json"
df3 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

features = ["Channel", "EventID", "Message", "DestPort", "LayerRTID", "
    ParentProcessName", "NewProcessName", "ParentImage", "Image", "DestinationPort
    ", "User"]
df3 = df3[features]
Y3 = labelList(df3.shape[0], [678, 478, 745, 856, 535, 867, 689, 1415, 1418])

gen_data3 = json.read_json(os.getcwd() + "\\generated_datasets\\
    empire_psremoting_stager_new1.json")

DNN_classifier(df3, Y3, [678, 478, 745, 856, 535, 867, 689, 1415, 1418],
    gen_data3, hidden_layers=5, manually=True)


# In[87]:
```

```
# Alternate PS
datasetJSONPath = os.getcwd() + "\\datasets\\empire_psremoting_stager_2020
    -09-20170827.json"
df4 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

features = ["Channel", "EventID", "Message", "Description", "ImageLoaded", "Image
    ", "PipeName"]
df4 = df4[features]
Y4 = labelList(df4.shape[0], [2, 4, 7, 8, 12, 768, 799])

gen_data4 = json.read_json(os.getcwd() + "\\generated_datasets\\
    empire_psremoting_stager_new2.json")

DNN_classifier(df4, Y4, [2, 4, 7, 8, 12, 768, 799], gen_data4, hidden_layers=5,
    manually=True)


# In[88]:


# Remote WMI
datasetJSONPath = os.getcwd() + "\\datasets\\
    empire_wmi_dcerpc_wmi_IWbemServices_ExecMethod_2020-09-21001437.json"
df5 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

features = ["Channel", "EventID", "ParentProcessName", "TargetLogonId", "
    LogonType", "SubjectUserName", "ParentImage", "LogonId"]
df5 = df5[features]
Y5 = labelList(df5.shape[0], [551, 368, 514])

gen_data5 = json.read_json(os.getcwd() + "\\generated_datasets\\
    empire_wmi_dcerpc_wmi_IWbemServices_ExecMethod_new.json")

DNN_classifier(df5, Y5, [551, 368, 514], gen_data5, hidden_layers=5, manually=
    True)


# In[90]:


# WMI Module
datasetJSONPath = os.getcwd() + "\\datasets\\empire_psinject_PEinjection_2020
    -08-07143205.json"
df6 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

features = ["Channel", "EventID", "ImageLoaded", "Image"]
df6 = df6[features]
Y6 = labelList(df6.shape[0], [2168, 2172, 2194, 2232, 2235])

gen_data6 = json.read_json(os.getcwd() + "\\generated_datasets\\
    empire_psinject_PEinjection_new.json")

DNN_classifier(df6, Y6, [2168, 2172, 2194, 2232, 2235], gen_data6, hidden_layers
    =5, manually=True)


# In[91]:


# Remote Installation
```

```
datasetJSONPath = os.getcwd() + "\\datasets\\empire_psexec_dcerpc_tcp_svcctl_2020
    -09-20121608.json"
df7 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

features = ["Channel", "EventID", "SubjectUserName", "SubjectLogonId", "
    TargetLogonId", "LogonType"]
df7 = df7[features]
Y7 = labelList(df7.shape[0], [1212, 1028])

gen_data7 = json.read_json(os.getcwd() + "\\generated_datasets\\
    empire_psexec_dcerpc_tcp_svcctl_new.json")

DNN_classifier(df7, Y7, [1212, 1028], gen_data7, hidden_layers=5, manually=True)
```

 # In[92]:

```
# Remote SCM Handle
datasetJSONPath = os.getcwd() + "\\datasets\\
    empire_find_localadmin_smb_svcctl_OpenSCManager_2020-09-22021559.json"
df8 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

features = ["Channel", "EventID","ObjectType", "ObjectName", "AccessMask", "
    SubjectLogonId", "PrivilegeList", "Application", "LayerRTID", "Image", "
    LogonType", "SubjectUserName", "TargetLogonId"]
df8 = df8[features]
Y8 = labelList(df8.shape[0], [1071, 1374, 854, 1066, 1100, 1372, 1373, 1374,
    1375, 1368, 1070, 1068])

gen_data8 = json.read_json(os.getcwd() + "\\generated_datasets\\
    empire_find_localadmin_smb_svcctl_OpenSCManager_new.json")

DNN_classifier(df8, Y8, [1071, 1374, 854, 1066, 1100, 1372, 1373, 1374, 1375,
    1368, 1070, 1068], gen_data8, hidden_layers=5, manually=True)
```

 # In[97]:

```
# Microphone
datasetJSONPath = os.getcwd() + "\\datasets\\msf_record_mic_2020-06-09225055.json
    "
df9 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

features = ["Channel", "EventID", "ObjectType", "ObjectName", "ProcessName", "
    SubjectLogonId", "ObjectValueName", "TargetObject"]
df9 = df9[features]
Y9 = labelList(df9.shape[0], [5117, 5118, 5202, 5203, 5204, 5240, 5241, 5242,
    5340, 5344])

gen_data9 = json.read_json(os.getcwd() + "\\generated_datasets\\
    msf_record_mic_new.json")

DNN_classifier(df9, Y9, [5117, 5118, 5202, 5203, 5204, 5240, 5241, 5242, 5340,
    5344], gen_data9, hidden_layers=5, manually=True)
```

 # In[93]:

```
# Active Script
datasetJSONPath = os.getcwd() + "\\datasets\\
    covenant_wmi_remote_event_subscription_ActiveScriptEventConsumers_2020
    -09-01214330.json"
df10 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

features = ["Channel", "EventID", "Message", "Image", "NewProcessName", "
    ImageLoaded", "LogonType", "ProcessName", "ProcessGuid", "ProcessId"]
df10 = df10[features]
Y10 = labelList(df10.shape[0], [157, 3307, 156, 1532, 1744, 1537, 1591, 1600,
    2151, 1799])

gen_data10 = json.read_json(os.getcwd() + "\\generated_datasets\\
    covenant_wmi_remote_event_subscription_ActiveScriptEventConsumers_new.json")


DNN_classifier(df10, Y10, [157, 3307, 156, 1532, 1744, 1537, 1591, 1600, 2151,
    1799], gen_data10, hidden_layers=5, manually=True)


# In[94]:


# Wbemcomn Hijack
datasetJSONPath = os.getcwd() + "\\datasets\\
    covenant_wmi_wbemcomn_dll_hijack_2020-10-09173318.json"
df11 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

features = ["Channel", "EventID", "RelativeTargetName", "SubjectUserName", "
    AccessMask", "Image", "TargetFilename", "ImageLoaded"]
df11 = df11[features]
Y11 = labelList(df11.shape[0], [450, 445, 651])

gen_data11 = json.read_json(os.getcwd() + "\\generated_datasets\\
    covenant_wmi_wbemcomn_dll_hijack_new.json")

DNN_classifier(df11, Y11, [450, 445, 651], gen_data11, hidden_layers=5, manually=
    True)


# In[95]:


# SMB Create
datasetJSONPath = os.getcwd() + "\\datasets\\covenant_copy_smb_CreateRequest_2020
    -09-22145302.json"
df12 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

features = ["Channel", "EventID", "ShareName", "SubjectUserName", "SubjectLogonId
    ", "AccessMask", "Image", "RelativeTargetName", "TargetFilename"]
df12 = df12[features]
Y12 = labelList(df12.shape[0], [58, 57, 61, 211])

gen_data12 = json.read_json(os.getcwd() + "\\generated_datasets\\
    covenant_copy_smb_CreateRequest_new.json")

DNN_classifier(df12, Y12, [58, 57, 61, 211], gen_data12, hidden_layers=5,
    manually=True)


# In[96]:
```

```
# Wuauclt
datasetJSONPath = os.getcwd() + "\\datasets\\
    covenant_lolbin_wuauclt_createremotethread_2020-10-12183248.json"
df13 = json.read_json(path_or_buf=datasetJSONPath, lines=True)

features = ["Channel", "EventID", "Image", "CommandLine", "Signed", "SourceImage"
    , "ImageLoaded", "TargetFilename", "SourceProcessGuid", "ProcessGuid"]
df13 = df13[features]
Y13 = labelList(df13.shape[0], [593, 612, 613, 87])

gen_data13 = json.read_json(os.getcwd() + "\\generated_datasets\\
    covenant_lolbin_wuauclt_createremotethread_new.json")

DNN_classifier(df13, Y13, [593, 612, 613, 87], gen_data13, hidden_layers=5,
    manually=True)


# ------------------------------------------------------------------------

# ### Normalization
#
# Z-score scaling: Standardize features by removing the mean and scaling to unit
    variance.
# $$x' = \frac{x-\mu} {\sigma} \$$
#

# In[29]:


# z-score scaling
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_train_new)
X_test = scaler.fit_transform(X_test)


# In[30]:


y_train = np.array(y_train)


# In[31]:


import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras.initializers import GlorotUniform
from tensorflow.keras.optimizers import Adam

# Define the input shape
input_shape = (8,)

# Neuron counts for the first hidden layer were three times the number of input
    neurons needed for the feature vector,
# The neuron counts for subsequent hidden layers were half od the preceding layer

# Define the number of neurons in each hidden layer
```

```
hidden_counts = [24, 12, 6, 3]

# Define the number of output neurons
num_classes = 1

# Define the activation function for the hidden layers
activation_fn = 'relu'

# Define the weight initializer
initializer = GlorotUniform(seed=42)

# Define the optimizer
optimizer = Adam()

# Define the model architecture
model = tf.keras.Sequential()

# Add the input layer
model.add(layers.InputLayer(input_shape=input_shape))

# Add the hidden layers
for i, count in enumerate(hidden_counts):
    model.add(layers.Dense(count, activation=activation_fn, kernel_initializer=
        initializer, name=f'hidden_{i+1}'))

# Add the output layer
# model.add(layers.Dense(num_classes, activation='softmax', kernel_initializer=
    initializer, name='output'))
model.add(layers.Dense(num_classes, activation='sigmoid', kernel_initializer=
    initializer, name='output'))


# Compile the model
# model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['
    accuracy'])
model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy
    ', 'AUC'])

hist = model.fit(X_train_new, y_train, verbose=2, epochs=50)
plt.plot(hist.history['accuracy'])
plt.show()


# In[34]:


# prediction
from sklearn import metrics

y_predict = model.predict(X_test)
y_predict_class = np.where(y_predict >= 0.5, 1, 0) # use 0.5 as threshold
draw_confusionMatrix(y_test, y_predict_class)


# ---------------------------------------------------------------------------
```