Joseph Allen (SUNetID: joeallen)
CS229 Spring 2021
Final Project Write-up

Title: Malware Classifier

Summary: I attempted to use XGBoost to use the random forest methodology to classify malware applications based on data extracted from static assembly files from known Android malware. The data set was found here https://www.unb.ca/cic/datasets/andmal2020.html. I spent a lot of time processing the data and loading it into a large CSV that consisted of about 357809 rows of data and 19043 features.

The malware categories

1) Adware
2) Backdoor
3) File Infector
4) PUA
5) Ransomware
6) Riskware
7) Scareware
8) Trojan
9) Trojan-Banker
10) Troajan—Dropper
11) Trojan-SMS
12) Trojan-Spy
13) Zero-day

In total there were 13 categories of malware, and then one large category for benign applications.

Here is a list of example features that are tracked for the dynamic data:

Memory_PssTotal,
Memory_PssClean,
Memory_SharedDirty,
Memory_PrivateDirty,
Memory_SharedClean,
Memory_PrivateClean,
Memory_SwapPssDirty,
Memory_HeapSize,
Memory_HeapAlloc,
Memory_HeapFree,
Memory_Views,
Memory_ViewRootImpl,
Memory_AppContexts,
Memory_Activities,
Memory_Assets,
Memory_AssetManagers,
Memory_LocalBinders,
Memory_ProxyBinders,
Memory_ParcelMemory,

Memory_ParcelCount,
Memory_DeathRecipients,
Memory_OpenSSLSockets,Memory_WebViews,
…
And there are thousands more.

What is interesting about the static data is that the features are not labelled. This is because the data already came from several rounds of preprocessing, so at this point we have lost the ability to make sense of the actual meaning of these features

The write-up on https://www.unb.ca/cic/datasets/andmal2020.html discusses the highlights of what type of features were used when coming up with the static analysis data. The AndroidManifest.xml had several features that they used including Activities, Broadcast receivers and providers, Metadata, the permissions requested and the System features.

Because all of these features were mashed together to produce the final CSV, the data will be used to classify the different executables, however I will not be able to make sense of what features went into play when making that decision. This is actually one of the things about machine learning that I find the most mind blowing, so I actually think that it's cool that this is possible to have a matrix with data that was mashed together from various features and it actually works in making predictions.

Method: The method that I decided to use was XGBoost to use the random forest techniques. The size of the file ended up giving me problems and the GCP instance I had ran out of memory. I have increased the memory and was hopeful to use late days to finish the project, but unfortunately I will have to do this on my own since I just read that late days aren't allowed.

Reason for Random Forest: I decided for the multi-class classifier for the 14 different categories based on static features from the assembly code that the random forest technique would be the best to find executables that were similar. I wouldn't expect there to be any real pattern and the same class of executable might have similarities with some in the class but than be quite different with others from the same class. The fact that random forest can handle those kinds messy patterns made it a good choice.

Where am I am now:
I have increased the size of the memory of my instance and am determined to keep running the XGBoost model over the next couple of days. I would greatly appreciate the chance to submit my results and see how well the XGBoost classifier performs.

I spent a great deal of time just figuring out how to compile the data that was spread across many files into one massive CSV file that is 7 GB big. I have a GCP instance with 30GB of memory now and I believe that I will get interesting results.

I intend to move forward and show the results I find in the non-graded poster.

Code:
The main code file to look at is train_model.py. It is pretty simple and uses XGBoost to run the classifier on my data.

This is the code:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
import xgboost
#import pickle
from joblib import dump
from joblib import load
from sklearn import model_selection
from sklearn.metrics import accuracy_score
from sklearn import preprocessing

import random
from datetime import datetime
random.seed(datetime.now())

input_file = "data/SUPER_TINY_SAMPLE_DATA.csv"


# comma delimited is the default
df = pd.read_csv(input_file, header = None)

print('CRONIN raw shape:')
print(df.shape)

df = df._get_numeric_data()
print('CRONIN numeric data only:')
print(df.shape)

dm = df.to_numpy()
print('CRONIN dm.shape')
print(dm.shape)

x = dm[:,1:]
y = dm[:,0:1]
le = preprocessing.LabelEncoder()
y = le.fit_transform(y)

print('CRONIN x.shape')
print(x.shape)

print('CRONIN y.shape')
print(y.shape)

x_train, x_test, y_train, y_test = train_test_split(x, y)

print('CRONIN x_train.shape')
```

```
print(x_train.shape)

print('CRONIN y_train.shape')
print(y_train.shape)


print('CRONIN x_test.shape')
print(x_test.shape)

print('CRONIN y_test.shape')
print(y_test.shape)


model = xgboost.XGBClassifier(
    use_label_encoder=False,
    learning_rate=0.1,
    n_estimators=1000,
    max_depth=5,
    min_child_weight=1,
    gamma=0,
    subsample=0.8,
    colsample_bytree=0.8,
    objective='multi:softmax',
    nthread=4,
    scale_pos_weight=1,
    seed=27,
    num_class=15,
    )


model.fit(x_train, y_train)

# make predictions for test data
y_pred = model.predict(x_test)
predictions = [round(value) for value in y_pred]
# evaluate predictions
accuracy = accuracy_score(y_test, predictions)
print("Accuracy: %.2f%%" % (accuracy * 100.0))
```

Right now, I have it running on a data set that is 10% the size of the primary data set, so there are about 30K rows of data from 30K different executables. I intend to press forward and get it using the 300k rows of data.