Week 7 - Lab 2 Ransomware Detection using K-NN

1 Introduction

Proliferation of cryptocurrencies (e.g., Bitcoin) that allow pseudo-anonymous transactions, has made it easier for ransomware developers to demand ransom by encrypting sensitive user data. The recently revealed strikes of ransomware attacks have already resulted in significant economic losses and societal harm across different sectors, ranging from local governments to health care. Most modern ransomware use Bitcoin for payments. However, although Bitcoin transactions are permanently recorded and publicly available, current approaches for detecting ransomware depend only on a couple of heuristics and/or tedious information gathering steps (e.g., running ransomware to collect ransomware related Bitcoin addresses).

In this tutorial, we would like to do Ransomware Detection on the Bitcoin Blockchain using the K-NN model.

2 Dataset

The dataset used is BitcoinHeist Ransomware Dataset

https://archive.ics.uci.edu/ml/datasets/BitcoinHeistRansomwareAddressDataset

Features

- address: String. Bitcoin address.
- year: Integer. Year.
- day: Integer. Day of the year. 1 is the first day, 365 is the last day.
- length: Integer.
- weight: Float.
- count: Integer.
- looped: Integer.
- neighbors: Integer.
- income: Integer. Satoshi amount (1 bitcoin = 100 million satoshis).
- label: Category String. Name of the ransomware family (e.g., Cryptxxx, cryptolocker etc) or white (i.e., not known to be ransomware).

Our graph features are designed to quantify specific transaction patterns. Loop is intended to count how many transaction i) split their coins; ii) move these coins in the network by using different paths and finally, and iii) merge them in a single address. Coins at this final address can then be sold and converted to fiat currency. Weight quantifies the merge behavior (i.e., the transaction has more input addresses than output addresses), where coins in multiple addresses are each passed through a succession of merging transactions and accumulated in a final address. Similar to weight, the count feature is designed to quantify the merging pattern. However, the count feature represents information on the number of transactions, whereas the weight feature represents information on the amount (what percent of these transactions' output?) of transactions. Length is designed to quantify mixing rounds on Bitcoin, where transactions receive and distribute similar amounts of coins in multiple rounds with newly created addresses to hide the coin origin.

White Bitcoin addresses are capped at 1K per day (Bitcoin has 800K addresses daily).

Note that although we are certain about ransomware labels, we do not know if all white addresses are in fact not related to ransomware.

When compared to non-ransomware addresses, ransomware addresses exhibit more profound right skewness in distributions of feature values.

3 Read the data

[1]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
```

[2]:
```
bitcoin_heist = pd.read_csv("BitcoinHeistData.csv")
```

4 Data Exploration

[3]:
```
bitcoin_heist.head()
```

[3]:
```
                              address  year  day  length    weight  count  \
0   111K8kZAEnJg245r2cM6y9zgJGHZtJPy6  2017   11      18  0.008333      1
1  1123pJv8jzeFQaCV4w644pzQJzVWay2zcA  2016  132      44  0.000244      1
2  112536im7hy6wtKbpH1qYDWtTyMRAcA2p7  2016  246       0  1.000000      1
3  1126eDRw2wqSkWosjTCre8cjjQW8sSeWH7  2016  322      72  0.003906      1
4  1129TSjKtx65E35GiUo4AYVeyo48twbrGX  2016  238     144  0.072848    456

   looped  neighbors       income            label
0       0          2  100050000.0  princetonCerber
1       0          1  100000000.0   princetonLocky
2       0          2  200000000.0  princetonCerber
3       0          2   71200000.0  princetonCerber
4       0          1  200000000.0   princetonLocky
```

[4]:
```
bitcoin_heist.describe()
```

[4]:
```
               year           day        length        weight         count  \
count  2.916697e+06  2.916697e+06  2.916697e+06  2.916697e+06  2.916697e+06
mean   2.014475e+03  1.814572e+02  4.500859e+01  5.455192e-01  7.216446e+02
std    2.257398e+00  1.040118e+02  5.898236e+01  3.674255e+00  1.689676e+03
min    2.011000e+03  1.000000e+00  0.000000e+00  3.606469e-94  1.000000e+00
25%    2.013000e+03  9.200000e+01  2.000000e+00  2.148438e-02  1.000000e+00
```

```
50%    2.014000e+03  1.810000e+02  8.000000e+00  2.500000e-01  1.000000e+00
75%    2.016000e+03  2.710000e+02  1.080000e+02  8.819482e-01  5.600000e+01
max    2.018000e+03  3.650000e+02  1.440000e+02  1.943749e+03  1.449700e+04

             looped     neighbors        income
count  2.916697e+06  2.916697e+06  2.916697e+06
mean   2.385067e+02  2.206516e+00  4.464889e+09
std    9.663217e+02  1.791877e+01  1.626860e+11
min    0.000000e+00  1.000000e+00  3.000000e+07
25%    0.000000e+00  1.000000e+00  7.428559e+07
50%    0.000000e+00  2.000000e+00  1.999985e+08
75%    0.000000e+00  2.000000e+00  9.940000e+08
max    1.449600e+04  1.292000e+04  4.996440e+13
```

[5]:
```python
bitcoin_heist.describe(include="O")
```

[5]:
```
                                   address    label
count                              2916697  2916697
unique                             2631095       29
top     1LXrSb67EaH1LGc6d6kWHq8rgv4ZBQAcpU    white
freq                                   420  2875284
```

[6]:
```python
bitcoin_heist.dtypes
```

[6]:
```
address       object
year           int64
day            int64
length         int64
weight       float64
count          int64
looped         int64
neighbors      int64
income       float64
label         object
dtype: object
```

[7]:
```python
bitcoin_heist
```

[7]:
```
                                    address  year  day  length    weight  \
0         111K8kZAEnJg245r2cM6y9zgJGHZtJPy6  2017   11      18  0.008333
1        1123pJv8jzeFQaCV4w644pzQJzVWay2zcA  2016  132      44  0.000244
2        112536im7hy6wtKbpH1qYDWtTyMRAcA2p7  2016  246       0  1.000000
3        1126eDRw2wqSkWosjTCre8cjjQW8sSeWH7  2016  322      72  0.003906
4        1129TSjKtx65E35GiUo4AYVeyo48twbrGX  2016  238     144  0.072848
…                                       …    …    …     …       …
2916692  12D3trgho1vJ4mGtWBRPyHdMJK96TRYSry  2018  330       0  0.111111
2916693  1P7PputTcVkhXBmXBvSD9MJ3UYPsiou1u2  2018  330       0  1.000000
```

```
2916694  1KYiKJEfdJtap9QX2v9BXJMpz2SfU4pgZw  2018  330       2  12.000000
2916695  15iPUJsRNZQZHmZZVwmQ63srsmughCXV4a  2018  330       0   0.500000
2916696  3LFFBxp15h9KSFtaw55np8eP5f6vdK17e   2018  330     144   0.073972

         count  looped  neighbors        income            label
0            1       0          2  1.000500e+08  princetonCerber
1            1       0          1  1.000000e+08   princetonLocky
2            1       0          2  2.000000e+08  princetonCerber
3            1       0          2  7.120000e+07  princetonCerber
4          456       0          1  2.000000e+08   princetonLocky
…          …       …          …           …               …
2916692      1       0          1  1.255809e+09            white
2916693      1       0          1  4.409699e+07            white
2916694      6       6         35  2.398267e+09            white
2916695      1       0          1  1.780427e+08            white
2916696   6800       0          2  1.123500e+08            white

[2916697 rows x 10 columns]
```

5 K-NN model for ransomware detection

[8]:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import f1_score
from sklearn import metrics
```

5.1 Convert categorical values to numeric values

Firstly, we convert a categorical column to numeric column: if a label is 'white', it is known to be ransomeware and is labelled 1. Otherwise, it is labelled 0

[9]:

```
bitcoin_heist["labels"] = [0 if x == 'white' else 1 for x in␣
↪bitcoin_heist['label']]
```

[10]:

```
bitcoin_heist["labels"].value_counts()
```

[10]:

```
0    2875284
1      41413
Name: labels, dtype: int64
```

5.2 Extract features

We use only the first 200000 instances. This step aims to reduce time complexity for identifing the optimal K. If we extract all instance, this will spend a lot of time.

[11]:
```
X = bitcoin_heist.loc[0:200000, ['year',"day", "length", "weight","count",␣
↪"looped", "neighbors", "income"]]
y = bitcoin_heist.loc[0:200000,'labels']
```

5.3 Split features and labels into a trainning and testing sets

[12]:
```
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    train_size = 0.8,
                                                    random_state=0)
```

5.4 Build a K-NN model

[13]:
```
model = KNeighborsClassifier(3)
```

5.5 Training the model and make a prediction

[14]:
```
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

5.6 Evaluate the model

[15]:
```
cm_knn = confusion_matrix(y_test, y_pred)
print(cm_knn)
```

```
[[31167   551]
 [ 1632  6651]]
```

[16]:
```
report_knn = classification_report(y_test, y_pred)
print(report_knn)
f1_knn = f1_score(y_test, y_pred,average='weighted')
print(f1_knn)
```

```
              precision    recall  f1-score   support

           0       0.95      0.98      0.97     31718
           1       0.92      0.80      0.86      8283

    accuracy                           0.95     40001
   macro avg       0.94      0.89      0.91     40001
weighted avg       0.94      0.95      0.94     40001

0.9439786835250186
```

5.7 Parameter Tunning using GridSearchCV

[17]:
```
from sklearn.model_selection import GridSearchCV
```

[18]:
```
model = KNeighborsClassifier()
```

[19]:
```
params = {'n_neighbors': range(1,10)}
```

[20]:
```
# 10-fold
#grs = GridSearchCV(model, param_grid=params, cv = 10)

# 5-fold default
grs = GridSearchCV(model, param_grid=params)
grs.fit(X_train, y_train)
```

[20]:
```
GridSearchCV(estimator=KNeighborsClassifier(),
             param_grid={'n_neighbors': range(1, 10)})
```

[21]:
```
print("Best Hyper Parameters:",grs.best_params_)
```

```
Best Hyper Parameters: {'n_neighbors': 3}
```

[22]:
```
y_pred=grs.predict(X_test)
```

[23]:
```
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print("Precision:",metrics.precision_score(y_test, y_pred, average =␣
 ↪'weighted'))
print("Recall:",metrics.recall_score(y_test, y_pred, average = 'weighted'))
print("F1-score:",metrics.f1_score(y_test, y_pred, average = 'weighted'))
```

```
Accuracy: 0.9454263643408914
Precision: 0.944703493081991
Recall: 0.9454263643408914
F1-score: 0.9439786835250186
```

Now, we will try to get more data to train the model

[24]:
```
X = bitcoin_heist[['year',"day", "length", "weight","count", "looped",␣
 ↪"neighbors", "income"]]
y = bitcoin_heist['labels']
```

[25]:
```
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    train_size = 0.8,
                                                    random_state=0)
```

[26]:
```
model = KNeighborsClassifier(3)
model.fit(X_train, y_train)
```

```
y_pred = model.predict(X_test)
```

[27]:
```
cm_knn = confusion_matrix(y_test, y_pred)
print(cm_knn)
```

```
[[573486   1648]
 [  6172   2034]]
```

[28]:
```
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print("Precision:",metrics.precision_score(y_test, y_pred, average =␣
 ↪'weighted'))
print("Recall:",metrics.recall_score(y_test, y_pred, average = 'weighted'))
print("F1-score:",metrics.f1_score(y_test, y_pred, average = 'weighted'))
```

```
Accuracy: 0.9865944389206981
Precision: 0.9832058580866595
Recall: 0.9865944389206981
F1-score: 0.9840699423444395
```