Piradon (Tien) Liengtiraphan
Project: Milestone
Instructor: Dr. Pablo Rivas

CMPT 404L-111: Artificial Intelligence
Due: 2016-12-12

Analysis of Failed Access

Solution.

Abstract

To see whether there is a relationship between attacks to protected resources on a college campus and methods of attack chosen, that can be identified using Unsupervised Machine Learning Algorithms; specifically analyzing cluster quality and significance using KMeans[1] and Spectral Clustering[3].

Introduction

The original idea behind the pursuit of this idea comes from my summer working in the IBM-Marist Joint Study. Each day the college receives an incredibly large number of attempted access, log-in attempts, extending all the way to DDoS attacks. The system is currently being protected by a Security System from BlackRidge technology, which keeps a log of all the "Discard" and "Forward" actions. These logs allow us to see where and how each rouge IP attempted to access the protected resource. Analyzing this data with a Machine Learning Algorithm might provide cyber-security professionals useful insight to the inner working of an attempted access and/or possible identify correlations that might have gone missing under the human eye. The Unsupervised Machine Learning Algorithm of K-Means was chosen to help identify possible correlations. This choice was made given the fact that we are currently uncertain whether a relationship exists or not, while over-fitting might become an issue, I believe it is first important to establish a base-case from which work can continue. The second algorithm chosen was Spectral Clustering, much like KMeans, Spectral Clustering allows us to find possible "clusters" or relations within the data provided, without knowing ahead of time whether such a relation exists or not.

Background and/or Related Work

I am uncertain if similar research has been done in this particular area, given that the data being used was collected and analyzed using tools that were written by students and the IBM-Marist Joint Study. However identifying pattern in and between cyber-security attacks is not a new area of studying. The usage of log files to dynamically update firewall setting is an avenue to researching being pursued by many in the industry. What I aim to achieve with this Machine Learning Project is to possible identify patterns in attacks as they coming in, allowing slightly delayed, if not real time identification of different attacks and/or attackers. As for how this project could be improved, more data from other systems should be collected, once that data is acquired we would run the Unsupervised Algorithm to see if the correlations found in this experiment hold true for multiple cases.

Methodology

The approach taken in this experiment is slightly different from what one might find if one had chosen a project from Kaggle. First and foremost this experiment focused on data collection, the analysis stage coming afterwards. This project has taken slightly longer than initially expected, due to the need for data collection and the code that has to be written to parse the data into a form that would be considered useful for analysis later on. Once the data was collected an appropriate Machine Learning Algorithm must be selected to analyze the data. As of last week we learned about Unsupervised Machine Learning Algorithms, specifically K-Means, given the nature of this experiment it seemed ideal to use K-Means due to its nature of finding possible correlations and relationship between data points that are unspecified[2]. Another Unsupervised Machine Learning Algorithm, that was not discussed in class, is Spectral Clustering, the ultimate goal being similar to K-Means, finding clusters within the data. The main difference between the two being Spectral Clustering leverages eigenvalues of the similarity Matrix of the data to perform dimensionality reduction[4]. Currently the data collected from the BlackRidge Boxes are formatted similarly to the 'features.csv' file provided in class, with 3 separate data point located in the first cell using a delimiter of a single white-space. Both K-Means and Spectral Clustering will be run against the same data set of Source_IP, Destination_IP, and Source_Port, located in the file 'IP-Port.csv', this will ensure the significance of the results of each algorithm when compared to each other. This data file will also be stored in a different folder to ensure that minimal manipulation of the data itself is made, if any at all.

Experiments

This data was obtained from the syslog of the 1G BlackRidge Authentication pair, which sits in the ECRL (Enterprise Computing Research Lab) in Hancock's Basement. The data was then parsed out using a script that I wrote to read the data and organize it in a way that was useful and usable for this particular experiment. The script leverages Python's re (Regular Expressions) library to identify and retrieve the data.

This data included: src_ip, src_port, dest_ip, dest_port, city, subdivision, lat, country, postal, long, host, host_name, isp_ip, asn. For usage in this experiment, given that the algorithms we learned cannot be used against data sets that are not numbers, each was assigned a unique value in decimal form to be used in this experiment. The experiment that will be performed on this particular data set will be different combinations of the data collected to see if a relationship and/or correlation and be found between the data points by the KMeans and Spectral Clustering Machine Learning Algorithms. Currently attempted analysis for a relationship between IP address and Source Port has begun. The first algorithm chosen to be run against the data was KMeans. The parameters chosen for the experiment were to run KMeans against the data set for a specified number of clusters. This helps us see at how many clusters did KMeans return the BestK. All these scores were recorded and then inputted into an array for later comparison. The range given to this particular run-through was between 0 and 100 clusters (full array located in reference pages). After the algorithm has finished running for the set number of clusters, we take the full array containing the BestK for each iteration and get the "best of the best" and its index. The score is representative of itself, while the index represent how many cluster(s) gave us the BestK. The experiment basis is replicated for and while using Spectral Clustering instead of KMeans, this is done to ensure significance when comparing the resulting data of both test algorithms (partial array located in reference pages).

Discussion and/or Analysis

With the help of Professor Rivas, each algorithm was modified to be compatible with the data set provided. In reference to the data achieved from the run through of each algorithm, it can be concluded that KMeans ran significantly better than Spectral Clustering. This is due to an bug in the code that results in Spectral Clustering throwing an error after 7 clusters. Given that a complete data set was not able to be dranw from Spectral Clustering. In an attempt to account for this error, when comparing the results of KMeans and Spectral Clustering, the data that will be reference will include everything up to 7 clusters, the point where Spectral Clustering ceased to work. When looking at the data generated from the KMeans algorithm, there is a clear lower bound for the number of clusters being between 0 and 100. The data never dropped below 0.3; however, the data also never went above 0.35 the BestK produced was a menial 0.347647249238, with the number of clusters being 1. What this means for the data is that there is a lack of discernible patterns within it, given that the best result of KMeans was generated when all the data was grouped into one large cluster. When looking at the the graph generated by the data (located in reference pages) we can see 3 discernible clusters and this is reflected in the final BestK Array. We can see that while the "best of the best" k returned was while the number of clusters was 1; however, closer inspection showed the second BestK produced was while the number of clusters was 3, which correlates to the graph generated. Next we shall examine the partial data acquired from the run-through of Spectral Clustering. While the algorithm did not successfully run, the general trend detect for each further iteration of the the algorithm with increasing cluster number resulted in a downwards slope for the "Spectral Score" returned. The best result from Spectral Analysis being 0.078790698768848838 when the number of clusters was 0. Analyzing this data allows us to conclude, as verified by two different algorithms that there truly is no discernible pattern located within the attack data provided. Recall, that this data comes only from IP and Source Port data, there may yet be discernible patterns to be located within other possible combinations of the collected data. Namely IP-Country, and IP-ASN; however, given the time constriction of the semester these data combination could not be tested as of yet. Comparing the two resulting array generated from the two algorithms respectively, it is safe to conclude that this particular data set has little to no discernible pattern; however, the extreme nature of the data maybe a result of not fully understanding the inner workings of Spectral Clustering and/or and incorrectly programed algorithm. In this case I believe that KMeans would be the better choice, simply off of the basis that we know it is correctly programmed and returned significant data. Should this project be continued, the next step would be to generate another combination of data to be run against KMeans and Spectral Clustering.

Conclusion

What I hope to achieve with this experiment was to be able to identify any discernible patterns located with cyber attack data. Ideally a correlation between the different data combinations can be found. This data will the be used to help identify specific attacks and/or sources of attacks to our system using pattern recognition from the data provided in this experiment. This pattern will hopefully be distinct enough to identify; however, not so unique to this particular use case that it is inapplicable to other similar environments. At the current time, near the conclusion of the semester, correlations within attack data, range from minimal to inconclusive. If we are to make an assumption based on the data that we have collected then the conclusion would have to be akin to: "As determined by KMeans and Spectral Clustering, there is no significant relationship to be detected within the attack data provided. Given that the returning BestK is 0.347647249238, which falls within the lower bounds of statistical significance. More data is necessary to make a conclusive conclusion to this particular problem set, there is a plethora of other possible data combination available that could not be tested within the given time frame. Once these have been thoroughly tested and Spectral Clustering verified to work properly will another conclusion be drawn." As stated in the interim conclusion, there are more data combinations to be tested, and only once all sets have been tested out will we be able to draw an overall conclusion. At the conclusion of this particular experiment I hope to possibly continue working and contributing to this experiment. This will allow me time to generate more data combinations and by extension, determine whether is are in fact discernible patters in attack data, as this information could aid future en devours for not only the college but cybersecurity practices. With this continuation new possibilities also open, give more time a more appropriate algorithm could possibly be found to better analyze the data, providing better data and enhanced interpretation of the data set provided, allowing for better conclusions to be drawn.

Reference Pages

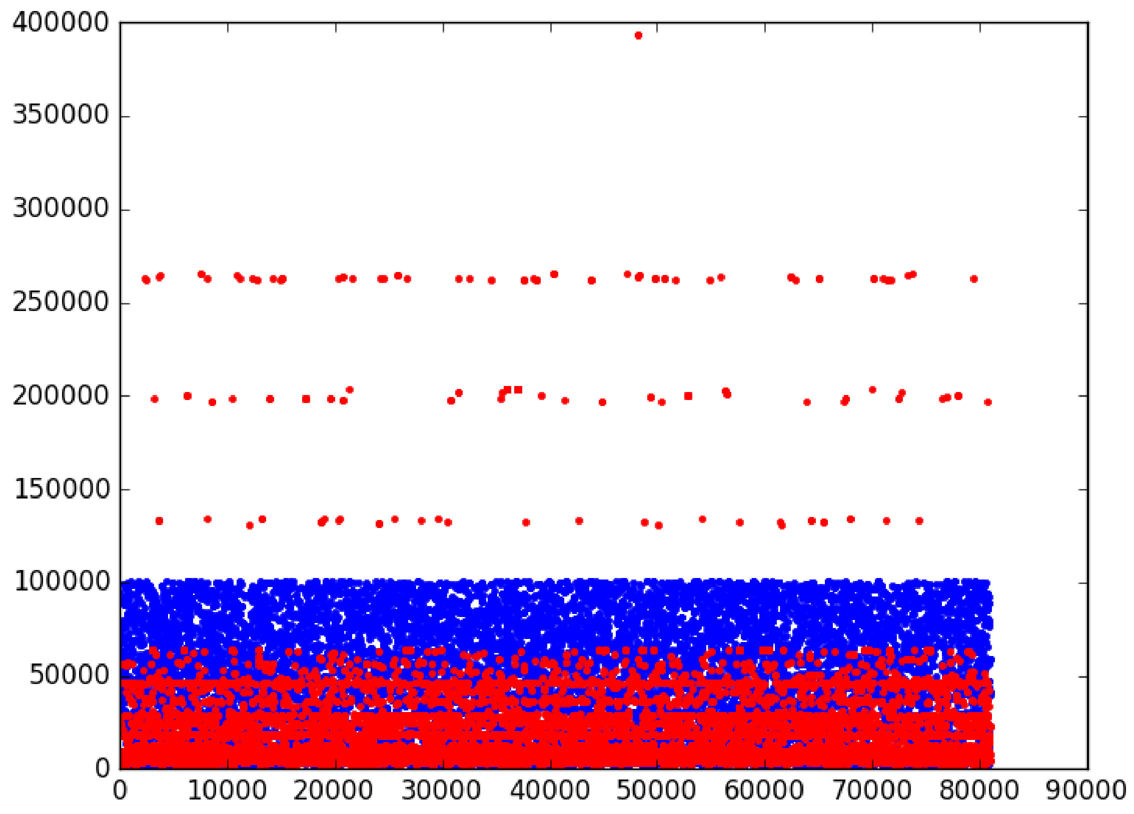
KMeans Results

```
============ BestK Array ============

[0.32275869091473375, 0.34764724923757118, 0.33866421104329725, 0.34638135390827729,
0.32575111171447835, 0.32848141292630906, 0.31315356058413479, 0.31832040622321156,
0.31648028791959371, 0.31904025245115764, 0.31960587720249939, 0.32054106620930595,
0.321129336027901, 0.31648087809478054, 0.31092805158523074, 0.31300520439377105,
0.3122994842662204, 0.30861379507265418, 0.30902109518610843, 0.30576496405350134,
0.30613192387169058, 0.30371027190526634, 0.29971218061186927, 0.30306239057784162,
0.30119913785882402, 0.3007101425460032, 0.30140303091032183, 0.30064108192011241,
0.30300110136761171, 0.30361844787607317, 0.3017514430777406, 0.30467726040380522,
0.30425299384868582, 0.30507847714856962, 0.30452802039669508, 0.30763357194013913,
0.30575129949681312, 0.30544633993126685, 0.30551563708807306, 0.30724674052460677,
0.30669594822270829, 0.30382837312105859, 0.30237644352314785, 0.30264603458452255,
0.30149185807483242, 0.30061814961556588, 0.30423920901840312, 0.30348836361071163,
0.30297605031088742, 0.30255252507053632, 0.3025018876054400 6, 0.30098832629245875,
0.30123208538526824, 0.30291883499670541, 0.30372504622638019, 0.30157338190586863,
0.30341007742686232, 0.30132208383240822, 0.30358136844704081, 0.30278541817900473,
0.30077679702321108, 0.30359355645257358, 0.30135321131581233, 0.3030482732066529,
0.30317255332048582, 0.3017559443130664, 0.30377457109315198, 0.30317976926709839,
0.3049207548742362, 0.30151074538680472, 0.3068646961685203, 0.30436423133928464,
0.30238066448126755, 0.3022251313436829, 0.30447080337784699, 0.30337073762178413,
0.30278897309021235, 0.30322161475723569, 0.30474378455332901, 0.30411644240281588,
0.30314404476675494, 0.30319297065094197, 0.3043519478599544, 0.30558587641600826,
0.30408363749237477, 0.30476981997861935, 0.30362591393998678, 0.30403345367546086,
0.30428300395103708, 0.30363918242214138, 0.30438168773539143, 0.30558286567039195,
0.30602840634334699, 0.30463085651286914, 0.30391930447846044, 0.30507362796999493,
0.30599399529457877, 0.30728476396143589, 0.30349791946531096]

============ BestK ============

0.347647249238

============ BestK Index ============

1
```

Spectral Clustering Results

```
============ Best Spectral Array ============

[0.078790698768848838, -0.068653104840236498, -0.19039401943456982, -0.21783814438250965, -
0.26770676796197179, -0.3549583782173325]

============ Best Spectral Score ============
0.0787906987688488838

============ Best Spectral Score Index ============

0
```

Generated Graph



KMeans.py

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn.cluster import KMeans
from sklearn import metrics
from sklearn.metrics import pairwise_distances
from numpy import genfromtxt
from sklearn.model_selection import KFold
import time
import os

# Start Timer
start_time = time.time()

# How many times to run KMeans.py
run_times = 1

for i in range(run_times):
    # read digits data & split it into X (training input) and y (target output)
    X = genfromtxt('Dataset/IP-Port.csv', delimiter=' ')

    plt.plot(X[:, 0], X[:, 1], '.')
    plt.plot(X[:, 0], X[:, 2], 'r.')
    # plt.show()

    bestk = []
    kc = 0
    result_num = len(os.listdir('KMeans_Results'))
    with open('KMeans_Results/kmeans_results' + str(result_num) + '.txt', 'w') as
file:
        for clusters in range(2, 101, 1):
            kf = KFold(n_splits=10)
            # clusters = 85
            kscore = []
            k = 0

            print("============ KScore ============")
            file.writelines("============ KScore ============" + "\n")
            for train, test in kf.split(X):
                #print("%s %s" % (train, test))
                X_train, X_test = X[train], X[test]

                #time.sleep(100)

                # we create an instance of Neighbors Classifier and fit the data.
                clf = KMeans(n_clusters=clusters)
                clf.fit(X_train)

                labels = clf.labels_
                kscore.append(metrics.silhouette_score(X_train, labels,
metric='euclidean'))

                print(kscore[k])
                file.writelines(str(kscore[k]) + "\n")
                k=k+1

            print("============ Clusters ============")
            file.writelines("============ Clusters ============" + "\n")
            print (clusters)
            file.writelines(str(clusters) + "\n")
            bestk.append(sum(kscore)/len(kscore))
            print("============ BestK[KC] ============")
            print(bestk[kc])
            file.writelines("============ BestK[KC] ============" + "\n")
            file.writelines(str(bestk[kc]) + "\n")
            kc+=1

            # to do here: given this array of E_outs in CV, find the max, its
            # corresponding index, and its corresponding value of clusters
            print("============ BestK Array ============")
            file.writelines("============ BestK Array ============" + "\n")
            print(bestk)
            file.writelines(str(bestk) + "\n")

        print("============ BestK ============")
        file.writelines("============ BestK ============" + "\n")
        print(max(bestk))
        file.writelines(str(max(bestk)) + "\n")
        print("============ BestK Index ============")
        file.writelines("============ BestK Index ============" + "\n")
        print(bestk.index(max(bestk)))
        file.writelines(str(bestk.index(max(bestk))) + "\n")

        # Calculates Running Time
        run_time = time.time() - start_time
        minutes, seconds = divmod(run_time, 60)
        hours, minutes = divmod(minutes, 60)
        print("============ Running Time ============")
        file.writelines("============ Running Time ============" + "\n")
        print("Seconds Time Format --- %s seconds ---" % (time.time() - start_time))
        print("Normal Time Format --- %d:%02d:%02d ---" % (hours, minutes, seconds))
        file.writelines(str("Seconds Time Format --- %s seconds ---" % (time.time() -
start_time)) + "\n")
        file.writelines(str("Normal Time Format --- %d:%02d:%02d ---" % (hours,
minutes, seconds)))
```

Spectral Clustering.py

```
import matplotlib.pyplot as plt
from sklearn import metrics
from numpy import genfromtxt
from sklearn.model_selection import KFold
from sklearn.cluster import SpectralClustering
import os
import time

# Start Timer
start_time = time.time()

# How many times to run KMeans.py
run_times = 1

for i in range(run_times):
    # read digits data & split it into X (training input) and y (target output)
    X = genfromtxt('Dataset/IP-Port.csv', delimiter=' ')

    plt.plot(X[:, 0], X[:, 1], '.')
    plt.plot(X[:, 0], X[:, 2], 'r.')
    # plt.show()

    bestk = []
    kc = 0
    result_num = len(os.listdir('KMeans_Results'))
    with open('Spectral_Clustering_Results/spectral_clustering_results' +
str(result_num) + '.txt', 'w') as file:
        for clusters in range(2, 101, 1):
            kf = KFold(n_splits=10)
            # clusters = 85
            kscore = []
            k = 0

            print("============ Spectral Score ============")
            file.writelines("============ Spectral Score ============" + "\n")
            for train, test in kf.split(X):
                #print("%s %s" % (train, test))
                X_train, X_test = X[train], X[test]

                #time.sleep(100)

                # we create an instance of Neighbors Classifier and fit the data.
                clf = SpectralClustering(n_clusters=clusters)
                clf.fit(X_train)

                labels = clf.labels_
                kscore.append(metrics.silhouette_score(X_train, labels,
metric='euclidean'))

                print(kscore[k])
                file.writelines(str(kscore[k]) + "\n")
                k=k+1

            print("============ Clusters ============")
            file.writelines("============ Clusters ============" + "\n")
            print (clusters)
            file.writelines(str(clusters) + "\n")
            bestk.append(sum(kscore)/len(kscore))
            print("============ Best Spectral[Spectral] ============")
            print(bestk[kc])
            file.writelines("============ Best Spectral[Spectral] ============" +
"\n")
            file.writelines(str(bestk[kc]) + "\n")
            kc+=1

            # to do here: given this array of E_outs in CV, find the max, its
            # corresponding index, and its corresponding value of clusters
            print("============ Best Spectral Array ============")
            file.writelines("============ Best Spectral Array ============" + "\n")
            print(bestk)
            file.writelines(str(bestk) + "\n")

        print("============ Best Spectral ============")
        file.writelines("============ Best Spectral ============" + "\n")
        print(max(bestk))
        file.writelines(str(max(bestk)) + "\n")
        print("============ Best Spectral Index ============")
        file.writelines("============ Best Spectral Index ============" + "\n")
        print(bestk.index(max(bestk)))
        file.writelines(str(bestk.index(max(bestk))) + "\n")

        # Calculates Running Time
        run_time = time.time() - start_time
        minutes, seconds = divmod(run_time, 60)
        hours, minutes = divmod(minutes, 60)
        print("============ Running Time ============")
        file.writelines("============ Running Time ============" + "\n")
        print("Seconds Time Format --- %s seconds ---" % (time.time() - start_time))
        print("Normal Time Format --- %d:%02d:%02d ---" % (hours, minutes, seconds))
        file.writelines(str("Seconds Time Format --- %s seconds ---" % (time.time() -
start_time) + "\n"))
        file.writelines(str("Normal Time Format --- %d:%02d:%02d ---" % (hours,
minutes, seconds)))
```

data_parser.py

```
import re
import xlwt
from tempfile import TemporaryFile

# Forwards Dataset
forward = "Dataset/forward_json.txt"
# Discard Dataset
discard = "Dataset/discard_json.txt"
# Test Dataset
test_data = "Dataset/test_json.txt"
# Debug Variable
debug = False

# Dataset Excel Name
'''data_sheet = xlwt.Workbook()
sheet1 = data_sheet.add_sheet("test")'''

# Print to Excel
'''def print_to_excel(info_list, sheet_name):
    for i,e in enumerate(info_list):
        sheet1.write(i,0,e)

    data_sheet.save(sheet_name)'''

# Function to grab data from json
def get_data(file_name):

    data = open(file_name).read()

    # Souce IP Information
    src_ip = re.findall('"src":\s"(\d+.\d+.\d+.\d+)"', data)
    src_port = re.findall('"src_port":\s"(\d+)"', data)

    # Destination IP Information
    dest_ip = re.findall('"dest":\s"(\d+.\d+.\d+.\d+)"', data)
    dest_port = re.findall('"dest_port":\s"(\d+)"', data)

    # Geolocation Information
    city = re.findall('"city":\s"(.*)",\s"host"', data)
    subdivision = re.findall('"subdivision":\s"(.*)",\s"name"', data)
    lat = re.findall('"lat":\s"(-?\d+.\d+)",\s"country"', data)
    country = re.findall('"country":\s"(.*)",\s"postal"', data)
    postal = re.findall('"postal":\s"(.*)",\s"ASN"', data)
    long = re.findall('"long":\s"(-?\d+.\d+)"}', data)

    # ISP Information
    host = re.findall('"host":\s"(.*)",\s"subdivision"', data)
    host_name = re.findall('"name":\s"(.*)",\s"ip"', data)
    isp_ip = re.findall('"ip":\s"(\d+.\d+.\d+.\d+)",\s"lat"', data)
    asn = re.findall('"ASN":\s"(\d+)",\s"long"', data)

    # Debug Messages
    if debug:
        print("src_ip: " + str(len(src_ip)))
        print("src_port: " + str(len(src_port)))
        print("dest: " + str(len(dest_ip)))
        print("dest_port: " + str(len(dest_port)))
        print("city: " + str(len(city)))
        print("subdivision: " + str(len(subdivision)))
        print("lat: " + str(len(lat)))
        print("long: " + str(len(long)))
        print("country: " + str(len(country)))
        print("postal: " + str(len(postal)))
        print("host: " + str(len(host)))
        print("host_name: " + str(len(host_name)))
        print("isp_ip: " + str(len(isp_ip)))
        print("asn: " + str(len(asn)))

    return src_ip, src_port, dest_ip, dest_port, city, subdivision, lat, country,
postal, long, host, host_name, isp_ip, asn

# Gets all the data and assigns it to the appropriate variable for printing later on
src_ip, src_port, dest_ip, dest_port, city, subdivision, lat, country, postal, long,
host, host_name, isp_ip, asn = get_data(discard)

#data_types = [src_ip, src_port, dest_ip, dest_port, city, subdivision, lat, country,
postal, long, host, host_name, isp_ip, asn]

def get_unique(list):
    ulist = set(list)

    '''for value in ulist:
        print(str(value))'''
    new_a = []

    for ip in ulist:
        new_a.append(ip)


    return new_a

#get_unique(src_port)
unique_list = get_unique(src_port)

for i in range(len(src_port)):
    for j in range(len(unique_list)):
        if src_port[i] == unique_list[j]:
            src_port[i] = j*10

for stuff in src_port:
    print(stuff)
#--------------------------------#
# Print Variables
src_ip_print = False
src_port_print = False
dest_ip_print = False
dest_port_print = False
city_print = False
subdivision_print = False
lat_print = False
country_print = False
postal_print = False
long_print = False
host_print = False
host_name_print = False
isp_ip_print = False
asn_print = False

if src_ip_print:
    for ip in src_ip:
        print(ip)
if src_port_print:
    for port in src_port:
        print(port)
```

```
if dest_ip_print:
    for ip in dest_ip:
        print(ip)
if dest_port_print:
    for port in dest_port:
        print(port)
if city_print:
    for city_name in city:
        print(city_name)
if subdivision_print:
    for subdivision_name in subdivision:
        print(subdivision_name)
if lat_print:
    for lat_num in lat:
        print(lat_num)
if country_print:
    for country_name in country:
        print(country_name)
if postal_print:
    for postal_num in postal:
        print(postal_num)
if long_print:
    for long_num in long:
        print(long_num)
if host_print:
    with open("temp.txt", "w") as file:
        for host_url in host:
            file.writelines(host_url+"\n")
if host_name_print:
    with open("temp.txt", "w") as file:
        for name in host_name:
            file.writelines(name + "\n")
if isp_ip_print:
    for ip in isp_ip:
        print(ip)
if asn_print:
    for num in asn:
        print(num)
#--------------------------------#
# Print All Records
'''for i in range(len(data_types)):
    for records in data_types[i]:
        print(records)'''
```

<u>Bibliography</u>

[1] Hartigan, J. A., and M. A. Wong. “Algorithm AS 136: A K-Means Clustering Algorithm.” *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, vol. 28, no. 1, 1979, pp. 100–108. www.jstor.org/stable/2346830.

[2] Hartigan, John A., and Manchek A. Wong. "Algorithm AS 136: A k-means clustering algorithm." *Journal of the Royal Statistical Society. Series C (Applied Statistics)* 28.1 (1979): 100-108.

[3] Ng, Andrew Y., Michael I. Jordan, and Yair Weiss. "On spectral clustering: Analysis and an algorithm." *Advances in neural information processing systems* 2 (2002): 849-856.

[4] Von Luxburg, Ulrike. "A tutorial on spectral clustering." *Statistics and computing* 17.4 (2007): 395-416.