THE UNIVERSITY OF MELBOURNE

COMP90073- Security Analytics

Assignment 2

Student Name: Lihua Wang
Student ID: 1164051
Tutor: Mark Jiang
Semester: 2020/S2

School of Computing and Information Systems

Table of Contents

Introduction

The organization structure of the report is as follows: In Task I, section 1 is an overview of the test dataset using Splunk, explaining feature generation/selection using SPL queries and Splunk native functionalities. Including the main features obtained by using regular Splunk query functions to find suspicious patterns and insights. The section 2 is the description of the methodology for generating features, which are based on literature review and the combination of Splunk query and python data manipulation and describes the method of generating and extracting features. The section 3 introduces the machine learning model to identify attacks the parameters and environment used in the experiment, as well as the scores and thresholds used to classify anomalies. In Task II, the section 1 explains the selected features and the supervised learning model used to train data. Section 2 is about describing the process of generating the adversarial samples to detect my model. In the conclusion part, it summarizes and discusses the methods of detecting network attacks, the comparison and analysis of model results, and gives conclusions.

Task I

1 Splunk Overview

1.1 Data Exploratory Analytics – Top Metrics

The raw dataset has 3 parts in CTU-13 NetFlow format, where training data denotes normal network traffic and test data denotes abnormal data containing botnet network traffic. I use Splunk dashboard to visualize some abnormal data and performed exploratory data analysis to determine the most relevant changes in network traffic patterns.

Splunk performed exploratory data analysis to determine the most relevant changes in network traffic patterns. In Figure 1.1, in the botnet, the TCP protocol accounts for most of the traffic, followed by common protocols such as ICMP and UDP. On the contrary, in the normal data set Figure 1.2, UDP is the most used protocol.

Top Protocol:

```
source="C:\\Program Files\\Splunk\\etc\\apps\\SplunkForPCAP\\PCAPcsv\\test_data.csv"
| stats count by protocol
| table protocol count
| sort – count
```

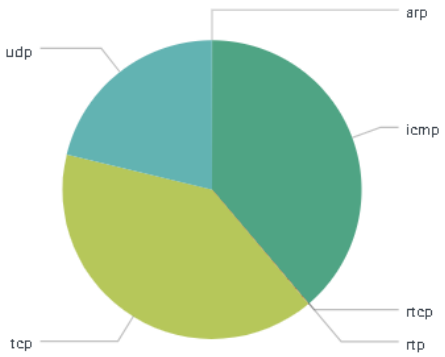

Figure 1.1 Top Protocols in Botnet Dataset

source="C:\\Program Files\\Splunk\\etc\\apps\\SplunkForPCAP\\PCAPcsv\\training_data.csv"

| stats count by protocol
| table protocol count
| sort - count

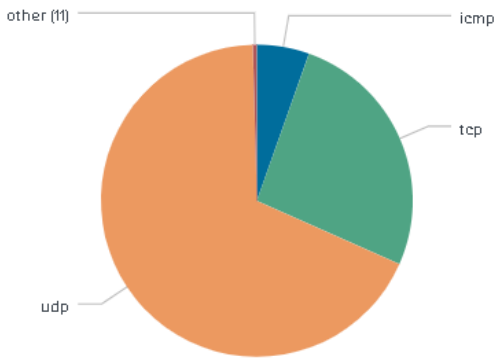

Figure 1.2 Top Protocols in Normal Dataset

Following this pattern, the figure 2.1 shows a significant change of proportion during the attack day compared to figure 2.2 showing the normal day port traffic.

Top Destination Ports:

source="C:\\Program Files\\Splunk\\etc\\apps\\SplunkForPCAP\\PCAPcsv\\test_data.csv"
| stats count by dst_port
| table dst_port count
| sort - count
| head 10

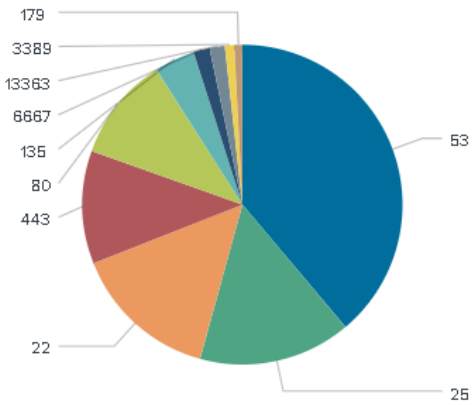

Figure 2.1 Top destination Port in Botnet dataset

source="C:\\Program Files\\Splunk\\etc\\apps\\SplunkForPCAP\\PCAPcsv\\training_data.csv"
| stats count by dst_port
| table dst_port count
| sort - count
| head 10

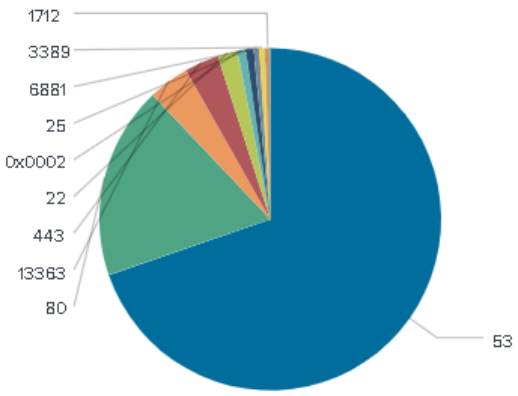

Figure 2.2 Top Destination Ports in Normal Dataset

Top src_ip:

```
source="C:\\Program Files\\Splunk\\etc\\apps\\SplunkForPCAP\\PCAPcsv\\test_data.csv"
| stats count by src_ip
| sort -count
| head 10
```

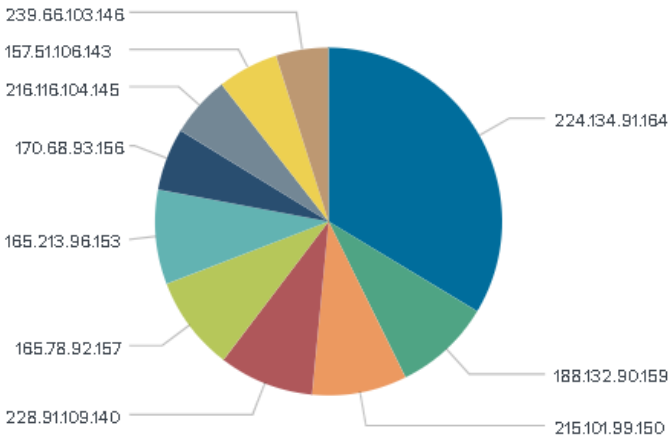

Figure 3 Top Source Ip in Botnet network

In the Botnet network, there are a significant amount of conversations involving external hosts with the host 125.189.87.2 as shown in Figure 4 which represents a destination from normal behavior.

Top dst_ip:

```
source="C:\\Program Files\\Splunk\\etc\\apps\\SplunkForPCAP\\PCAPcsv\\test_data.csv"
| stats count by dst_ip
| sort -count
| head 10
```

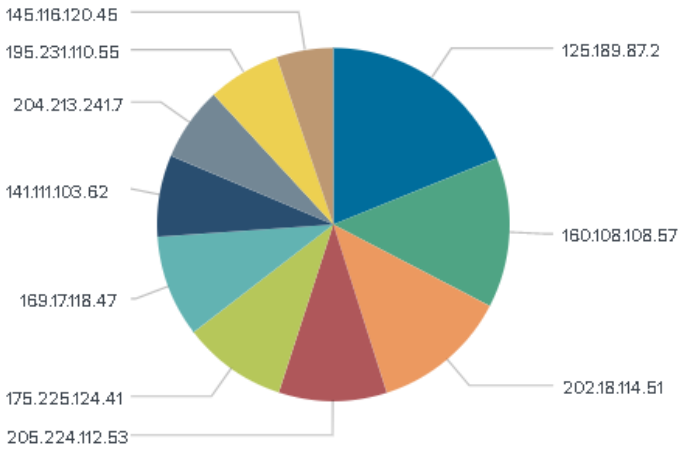

Figure 4 Top Destination Ip in Botnet network

The proportion of the number of conversations with the host should also be highlighted. Obviously, several hosts have been sending and receiving data packets for conversations with the host at a similar ratio. May be attacked by DoS (DDoS).

Top Conversations:

```
source="C:\\Program Files\\Splunk\\etc\\apps\\SplunkForPCAP\\PCAPcsv\\test_data.csv"
| eval conversation=src_ip+" -> "+dst_ip
| stats count by conversation
| table conversation count
| dedup conversation
| sort -count
| head 10
```

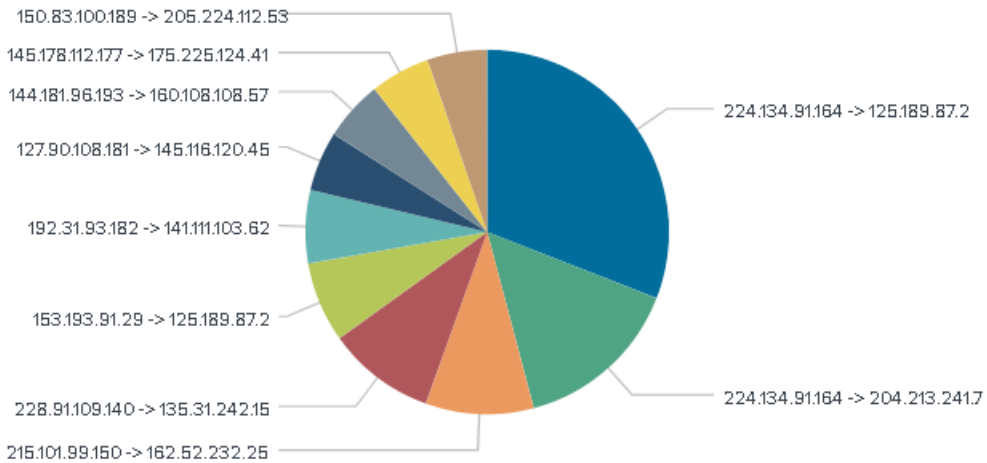

Figure 5.1.1 Top Conversations in Botnet Dataset

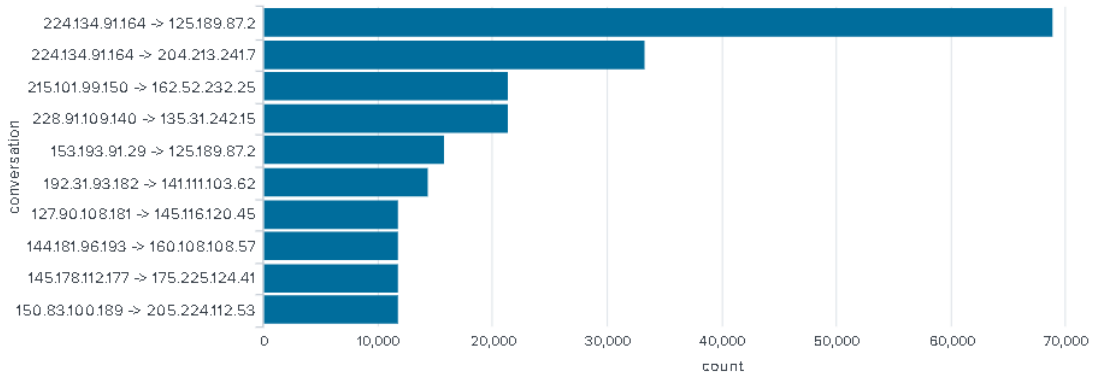

Figure 5.1.2 Top Conversations in Botnet Dataset (Numbers)

```
source="C:\\Program Files\\Splunk\\etc\\apps\\SplunkForPCAP\\PCAPcsv\\test_data.csv"
| eval conversation=src_ip+" -> "+dst_ip
| stats count by conversation
| table conversation count
| dedup conversation
| sort -count
| head 10
```

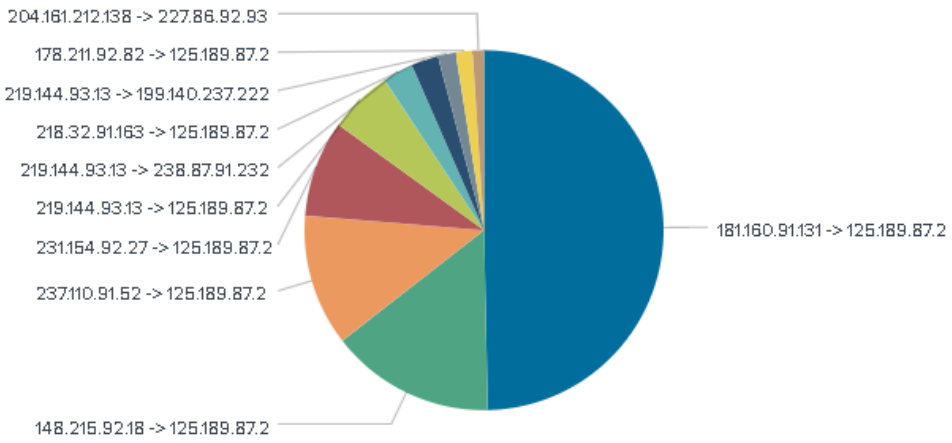

Figure 5.2 Top Conversations in Normal dataset

Port Scan:

```
source="C:\\Program Files\\Splunk\\etc\\apps\\SplunkForPCAP\\PCAPcsv\\test_data.csv"
| timechart span=1h dc(dst_port) by src_ip usenull=f
```

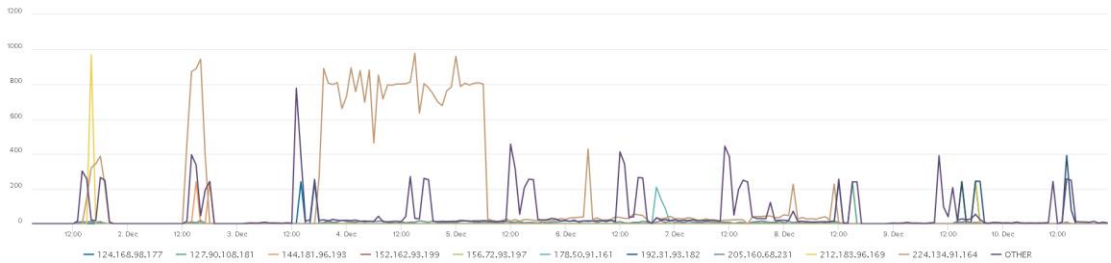

Figure 6 Port Scan in botnet network

TotPkts transferred by src_ip:

```
source="C:\\Program Files\\Splunk\\etc\\apps\\SplunkForPCAP\\PCAPcsv\\test_data.csv"
| timechart span=1h dc (TotPkts) by src_ip usenull=f
```

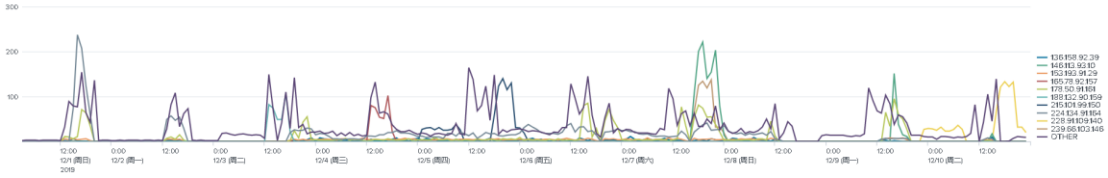

Figure 7 TotPkts transferred by src_ip in botnet network

TotBytes transferred by src_ip:

source="C:\\Program Files\\Splunk\\etc\\apps\\SplunkForPCAP\\PCAPcsv\\test_data.csv"
| timechart span=1h dc (TotBytes) by src ip usenull=f

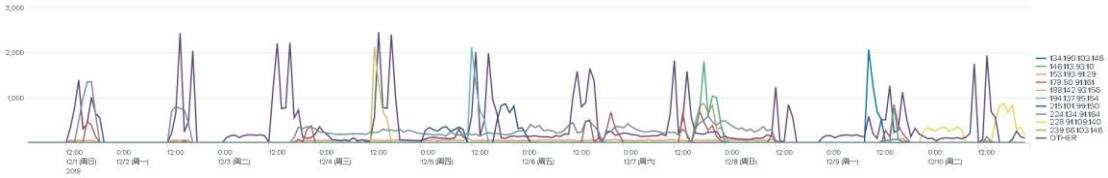

Figure 8 TotBytes transferred by src_ip in Botnet network

TotBytes transferred by dst_ip:

source="C:\\Program Files\\Splunk\\etc\\apps\\SplunkForPCAP\\PCAPcsv\\test_data.csv"
| timechart span=1h dc(TotBytes) by dst_ip usenull=f

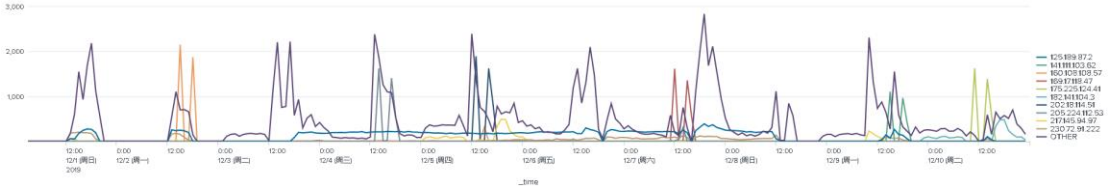

Figure 9 TotBytes transferred by dst_ip in Botnet network

1.2 Feature Analytics

I made some basic exploration to find the correlation in the basic features' dataset {duration, sTos, dTos, TotPkts, TotBytes, SrcBytes}, which can help apply the cluster analysis functions.

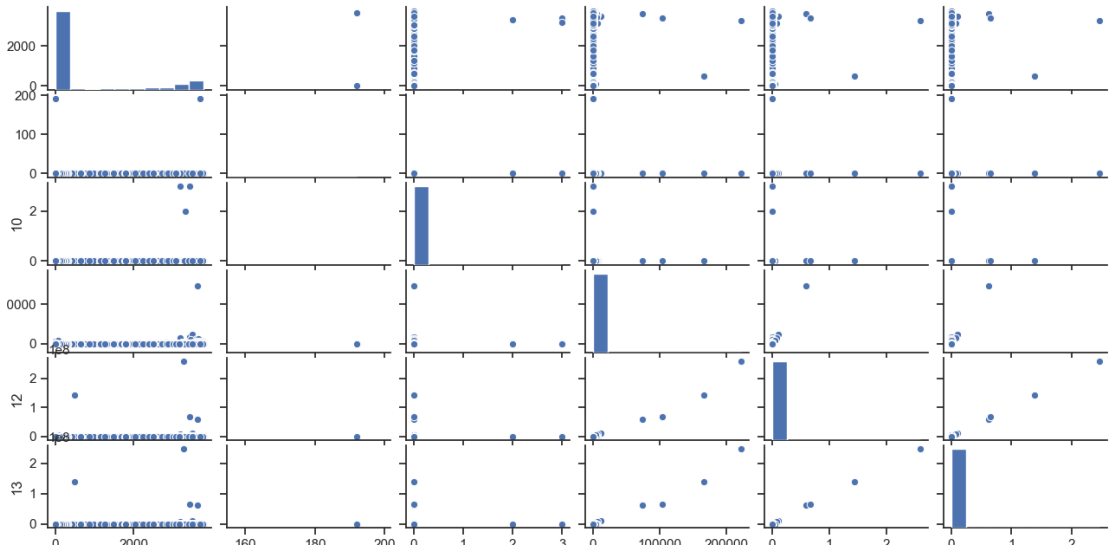

Figure 10 Features correlation

Overall, the test dataset shows clear deviations in the following aspects:

- Traffic between hosts,
- Transferred data,
- Port used,
- Protocol used
- Conversation intensive

Considering this deviation, I created a set of features that can be used in existing ML models to describe attacks that can be triggered by deviation-related features sufficient to identify anomalous data. The first feature mode was generated by SPL, while the rest is generated using python's transformations and more complex data processing algorithms.

Feature sets fall into 3 categories:

- Basic functionality: Collects from data packet headers such as protocol type, service, TCP flags, source and destination flags.
- Time-based functions: An example is the amount of data transferred during a particular window period.
- ML-based functions: Using One-hot encoding to find the state, protocol and direction relations.

2 Feature Generation/Extraction

2.1 Using Splunk (Feature 1):

First extract new fields by using splunk internal function to index each column.

Figure 11 Extract new fields in Splunk

Next, rename each field as timestamp, duration, protocol, src_ip, src_port, direction, dst_ip dst_port TotPkts TotBytes SrcBytes.

For feature mode 1, I extracted some new features based on time where number of data transferred during a time window. Like pps, pbs, bpp, calculated the speed of transferred bytes and packets per second, since Botnet network flow usually send many requests in a short time.

Then extract the features from test and train datasets by using SPL and exported the modified datasets as the first feature mode.

SPL:

```
source="C:\\Program Files\\Splunk\\etc\\apps\\SplunkForPCAP\\PCAPcsv\\test_data.csv"
| eval pps = TotPkts/duration
| eval bps = TotBytes/duration
| eval bpp = TotBytes/TotPkts
| table duration protocol src_ip src_port dst_ip dst_port TotPkts TotBytes SrcBytes pps bps bpp
```

duration	protocol	src_ip	src_port	dst_ip	dst_port	TotPkts	TotBytes	SrcBytes	pps	bps	bpp
9.067028	tcp	157.51.106.143	1867	226.90.25.126	22	3	177	180	0.3308692	19.52128	59
1.205164	tcp	228.91.109.140	6060	135.31.242.15	443	7	336	181	5.808338	278.8002	48
8.23631	tcp	157.51.106.143	1865	140.18.193.101	22	3	181	193	0.364241	21.9759	60.333333333333336
9.353172	tcp	157.51.106.143	1864	162.58.154.76	22	4	187	172	0.4276624	19.99322	46.75
1.316382	tcp	228.91.109.140	6059	135.31.242.15	443	7	374	193	5.317605	284.1121	53.42857142857143
8.996432	tcp	157.51.106.143	1863	236.145.32.139	22	5	200	178	0.5557759	22.23104	40

Figure 12 Feature mode 1 selected by Splunk

Feature	Type	Description
Numeric Features		
duration	Number	Total in seconds
protocol	String	The protocol used in the conversation
src_ip	String	Source IP
src_port	String	Source Port
dst_ip	String	Destination IP
dst_port	String	Destination Port
TotPkts	Number	the number of total packets
TotBytes	Number	the number of bytes transferred in both directions
SrcBytes	Number	the number of bytes transferred from the source to the destination
pps	Number	Number of Packets per second
bps	Number	Number of bytes per second
bpp	Number	Number of bytes per packet

Table 1 Feature mode 1 (Selected by Splunk)

2.2 Using Python (Feature 2&3):

2.2.1 One-hot feature engineering

I apply the one-hot encoding based on the first feature mode and to process some 'String' type features ('protocol', 'direction', 'state'). First, I count the frequencies of different types in each feature, and only keep the top N frequency types, the other types were all classified as 'other'.

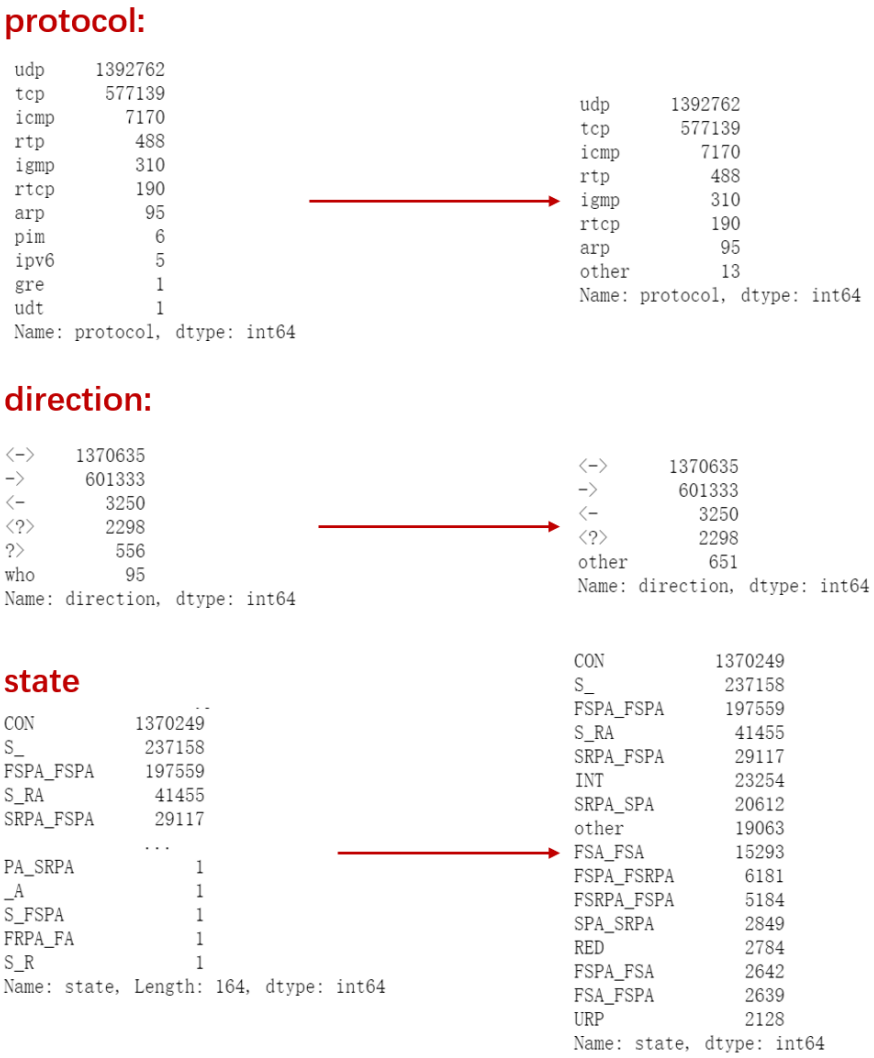

Figure 13 One-hot Encoding process

Then, apply one-hot feature engineering and the feature mode 2 were shown as Table 2.

Feature	Type	Description
Numeric Features		
duration	Number	Total in seconds
protocol	String	The protocol used in the conversation
src_ip	String	Source IP
src_port	String	Source Port
dst_ip	String	Destination IP
dst_port	String	Destination Port
TotPkts	Number	the number of total packets
TotBytes	Number	the number of bytes transferred in both directions
SrcBytes	Number	the number of bytes transferred from the source to the destination
pps	Number	Number of Packets per second
bps	Number	Number of bytes per second
bpp	Number	Number of bytes per packet
One-hot encoding based on numeric features		

protocol_udp	Number	The protocol udp used in the conversation
protocol_tcp	Number	The protocol tcp used in the conversation
protocol_icmp	Number	The protocol icmp used in the conversation
protocol_rtp	Number	The protocol rtp used in the conversation
protocol_igmp	Number	The protocol igmp used in the conversation
protocol_rtcp	Number	The protocol rtcp used in the conversation
protocol_arp	Number	The protocol arp used in the conversation
protocol_other	Number	Other protocols used in the conversation
direction_<->	Number	Bi-direction in the conversation
direction_->	Number	Uni-direction (src->dst) in the conversation
direction_<-	Number	Uni-direction (dst->src) in the conversation
direction_<?>	Number	Not clear the direction in the conversation
direction_other	Number	Other types of direction
state_CON	Number	Connection flags to show connection detail: CON
state_S_	Number	Connection flags to show connection detail: S
state_FSPA_FSPA	Number	Connection flags to show connection detail: SFSPA_FSPA
state_S_RA	Number	Connection flags to show connection detail: S_RA
state_SRPA_FSPA	Number	Connection flags to show connection detail: SRPA_FSPA
state_INT	Number	Connection flags to show connection detail: INT
state_SRPA_SPA	Number	Connection flags to show connection detail: SRPA_SPA
state_FSA_FSA	Number	Connection flags to show connection detail: FSA_FSA
state_other	Number	Other connection flags

Table 2 Feature mode 2 (Selected by One-hot Encoding)

There is reference about state flags representation [1].

```
ASA5515-X# show conn detail
35 in use, 199 most used
Flags: A - awaiting inside ACK to SYN, a - awaiting outside ACK to SYN,
       B - initial SYN from outside, b - TCP state-bypass or nailed,
       C - CTIQBE media, c - cluster centralized,
       D - DNS, d - dump, E - outside back connection, F - outside FIN, f - inside FIN,
       G - group, g - MGCP, H - H.323, h - H.225.0, I - inbound data,
       i - incomplete, J - GTP, j - GTP data, K - GTP t3-response
       k - Skinny media, M - SMTP data, m - SIP media, n - GUP
       O - outbound data, P - inside back connection, p - Phone-proxy TFTP connection,
       q - SQL*Net data, R - outside acknowledged FIN,
       R - UDP SUNRPC, r - inside acknowledged FIN, S - awaiting inside SYN,
       s - awaiting outside SYN, T - SIP, t - SIP transient, U - up,
       V - VPN orphan, W - WAAS,
       X - inspected by service module,
       x - per session, Y - director stub flow, y - backup stub flow,
       Z - Scansafe redirection, z - forwarding stub flow
```

Figure 14 The flags represented in state fields

2.2.2 One-hot + PCA feature engineering

After one-hot process, there are too many features and may cause curse of dimensionality, this is because the higher the dimension, the sparser the distribution of data on each characteristic dimension. So, it's important to scale variables and

dimensionality reduction was performed using PCA, which produce some features that explain most of the variability of the data.

However, we need to find the best reduced dimensionalities when applying PCA, so Using the PCA method encapsulated by *sklearn* library in python. The PCA method parameter *n_components*, if it is set to a decimal, it means the information that the reduced data can retain. At this time, we choose 0.9 which can keep more than 90% data after reducing the dimensions.

In the end I reduced dimensionalities into 4 measures {PC_1, PC_2, PC_3, PC_4}.

PC_1	PC_2	PC_3	PC_4
34689.28040000476	12302.738857370161	-3726.935836128255	112.25852691712271
-42237.22781766315	-2748.6274764049085	57.54350957491714	5.827471550086223
-42107.66033105271	-2695.873359655983	-3170.084615571266	-462.79811934180265
-38753.80761712594	-2172.600094744168	-2881.9214208568724	-402.50678335239223
-40572.00607394015	-2508.4460703710683	-3256.193801095173	-468.64719885880226
-41947.16611559894	-2697.078958544611	-627.1821447096943	-92.44789629312024
-39704.27729465686	-2470.51752998355	-3178.1940322389696	-454.943453195791
-42455.86587960373	-2799.6488320306526	57.801382733045486	4.051903081824413
-42389.99711707058	-2760.937001504215	57.52387659811386	5.955397367335273
-40485.04080102888	-2512.215440877968	-3447.22476627794	-496.6224184165942

Figure 15 One-hot & PCA Feature Engineering

2.3 Feature Methods Comparison between Project 1&2

In Project 1, I just used some basic features which were selected by intuition based on the results of exploratory analysis. The continuous increase in traffic involving two specific hosts and the amount of data transferred led to the conclusion of a DoS attack. The anomaly detection method used helps to identify anomalies based on extreme values, but since only a few functions are used, by comparing the results with the previous analysis, many false positives are finally discarded.

For Project 2, it is considered to be a more complex set of functions, including not only basic functions, but also time-based functions and data dimensions processed. Two methods are used to generate feature. Compared to the query used in the first project, the query shown in Table 2 generates the dialogue in a more fine-grained manner.

3 Anomaly Detection

I build the anomaly detection models using Python and import the library sklearn to help me fit the different models, where has already encapsulated functions of clustering algorithms.

3.1 OPTICS (DBSCAN)

I have tried to fit the DBSCAN model, however, due to the computational resources limitations, it wasted much time and cannot run to a result. So, I chose a similar method to implement, that is, using OPTICS (Ordering Points to Identify the Clustering Structure) to find core samples of high density and expands clusters from them. OPTICS creates a data sequence that represents a density-based clustering structure. Then extract the cluster using a method similar to DBSCAN.

3.1.1 Experimental Setup

Similar to the DBSCAN algorithm, OPTICS has two important parameters: *Eps* and *MinPts*. The former is the neighbourhood radius when the density is defined, and the latter is defined as the minimum number of neighbours owned by the core point. We first classify the data points, the number of data points in the neighbourhood: $\rho(x) \geq MinPts$, which we call core points. The non-core points in the neighbourhood of the core point we become boundary points, and the others are noise points and are not in the neighbourhood of any core point. Due to time constraints and computing resource constraints, these parameters have not been adjusted.

First, determine the *Eps* value. The Eps value will affect the number of outliers identified by the algorithm. If it is too small, most of the data will not be marked in the cluster and will be considered as outliers. Conversely, if the value is too high, the clusters will be merged and the data points will fall within the clusters. This parameter affects the threshold for identifying data points as outliers. This time set Eps=5 as the first attempt.

Then determine *MinPts*, there is a recommendation to define a value larger than the dimension (P): "Minimum number of samples"> "P". Therefore, we set the minimum sample number to 54.

3.1.2 Evaluation (Scoring and threshold technique)

The score is derived from the parameters used to run the OPTICS clustering algorithm. As mentioned in the experimental settings, Eps=5, the smallest sample that is close to the mean distance and larger than the number of parameters, affects how the algorithm classifies data points as outliers: $cluster = 1$.

Note: Some experiments have been done using this algorithm, but no results are displayed because there is no final output for the latest feature set.

3.2 Isolation Forest (IF)

The IForest algorithm is suitable for the detection of anomalies in continuous numerical data. Anomalies are defined as points that are sparsely distributed and far away from high-density groups. In *sklearn*, we use Isolation Forest in the ensemble package for outlier detection. First, the features are randomly selected, and then the split values between the maximum and minimum values of the selected features are randomly selected to separate the observations. This method does not use the concept of distance described in the previous method because it attempts to isolate the anomaly instead of analysing the usual data points.

3.2.1 Experimental Setup

iForest establishes an iTree collection and identifies anomalies as short average path length instances on iTree. In this analysis, only one parameter is passed to the algorithm and the rest remain the default values implemented by sklearn. This parameter represents the outlier ratio and is used to define the threshold for the decision function. To simplify the analysis, the parameter is set to 1% by default, taking into account the size of the test dataset.

3.2.2 Evaluation (Scoring and threshold technique)

Thresholds are determined by contam parameters that affect decision-making function. For instances identified as outliers, the score is -1, otherwise, it is +1.

Establishing a threshold equal to the topmost frequent 15 conversations in the anomalies results.

Results:

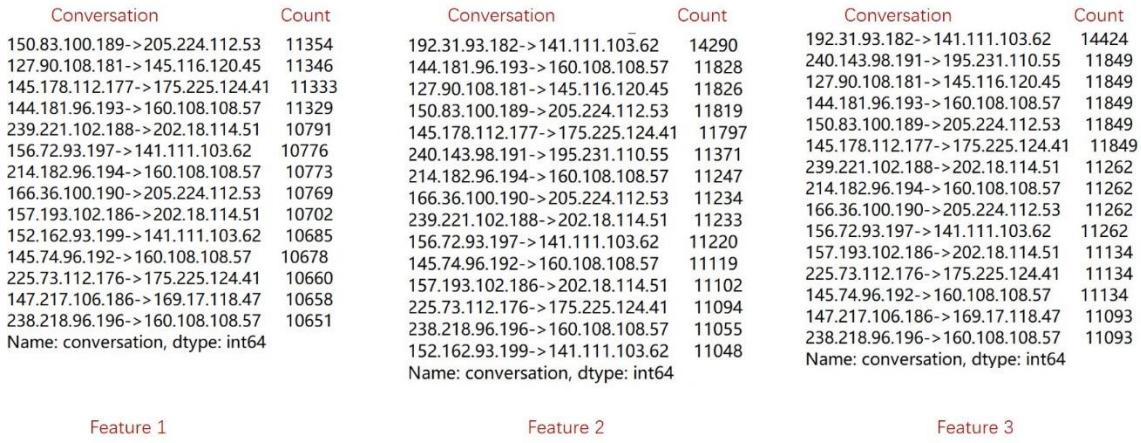

Figure 16 IForest Results Applying in the Test Data

As the results shown above, there are top 15 frequencies of conversation based on three features, we can find that host 145.116.120.45 might be victim host.

IForest Score in test data:

Type	Feature 1	Feature 2	Feature 3
TN	0.046	0.052	0.046
FP	0.017	0.011	0.016
FN	0.623	0.626	0.621
TP	0.314	0.311	0.316
TPR	**0.335**	**0.332**	**0.337**
FPR	**0.264**	**0.178**	**0.257**
mark	**71.052**	**32.993**	**63.188**
Precision	0.950	0.965	0.951
Recall	0.335	0.332	0.337
F1_score	0.496	0.494	0.498
ACC	0.361	0.363	0.362

AUC_score	0.536	0.577	0.540

Table 3 IForest Evolution Results in Test Data

Compare with the three features, the Feature mode 1 has a little better result that the TPR and FPR score denotes a higher grade. However, the whole IForest model didn't fit well to detect the abnormal data in test dataset, because all three features' TPR and FPR scores contribute a really low mark.

3.3 Local Outlier Factor (LOF)

In many cases, outliers do not have binary characteristics, that is to say, it is meaningful to explore the degree of separation between them and the surrounding neighbourhood. Assign an abnormality level value to each object, which is the local abnormality factor (LOF). For the object in the centre of the data set, the LOF value is approximately 1. For other objects, regardless of whether MinPts-nearest neighbours come from one or more data sets, LOF values with strict lower and upper limits are given.

3.3.1 Experimental Setup

Similar to previous density-based algorithms, LOF is based on the parameter *MinPts* and is used to identify the number of nearest neighbours of an instance's local neighbours. Determine the number of neighbours using a method similar to the OPTICS algorithm. $MinPts > P$, where P is the number of dimensions in the dataset.

3.3.2 Evaluation (Scoring and threshold technique)

Outliers are determined by instances that have a lower density than adjacent instances. The value -1 is assigned to each instance classified as an outlier. Otherwise, 1 is assigned. Many heuristics for choosing the value of the parameter MinPts, which affects the thresholds of the algorithm for classifying outliers, but most of them are for choosing a reasonable value.

Results:

Feature 1

Conversation	Count
140.96.100.191->205.224.112.53	373
239.221.102.188->202.18.114.51	357
157.193.102.186->202.18.114.51	346
150.83.100.189->205.224.112.53	332
183.203.100.193->205.224.112.53	328
192.31.93.182->141.111.103.62	324
232.48.94.183->127.57.106.59	313
133.171.104.189->214.220.116.49	300
127.90.108.181->145.116.120.45	295
168.188.94.155->127.57.106.59	290
203.174.102.174->202.18.114.51	287
172.227.102.189->202.18.114.51	266
145.178.112.177->175.225.124.41	262
145.74.96.192->160.108.108.57	260

Feature 2

Conversation	Count
166.36.100.190->205.224.112.53	950
192.31.93.182->141.111.103.62	893
150.83.100.189->205.224.112.53	808
239.221.102.188->202.18.114.51	796
140.96.100.191->205.224.112.53	779
157.193.102.186->202.18.114.51	746
232.48.94.183->127.57.106.59	728
127.90.108.181->145.116.120.45	713
183.203.100.193->205.224.112.53	698
168.188.94.155->127.57.106.59	695
145.178.112.177->175.225.124.41	683
170.166.102.191->202.18.114.51	672
225.73.112.176->175.225.124.41	665
191.138.112.181->175.225.124.41	662
172.227.102.189->202.18.114.51	645

Name: conversation, dtype: int64

Feature 3

Conversation	Count
166.36.100.190->205.224.112.53	830
192.31.93.182->141.111.103.62	782
239.221.102.188->202.18.114.51	673
150.83.100.189->205.224.112.53	671
140.96.100.191->205.224.112.53	656
157.193.102.186->202.18.114.51	640
127.90.108.181->145.116.120.45	588
232.48.94.183->127.57.106.59	581
183.203.100.193->205.224.112.53	576
170.166.102.191->202.18.114.51	565
168.188.94.155->127.57.106.59	551
224.134.91.164->125.189.87.2	539
172.227.102.189->202.18.114.51	529
203.174.102.174->202.18.114.51	528
156.72.93.197->141.111.103.62	527

Name: conversation, dtype: int64

Figure 17 LOF Results Applying in the Test Data

As the results shown above, there are top 15 frequencies of conversation based on three features, we can find that 150.83.100.189 might be a botnet host to attack host 205.224.112.53 (victim).

LOF Score in test data:

Type	Feature 1	Feature 2	Feature 3
TN	0.06267	0.06266	0.06264
FP	0.00095	0.00023	0.00038
FN	0.93658	0.93602	0.9359
TP	0.00072	0.00128	0.00132
TPR	**0.00077**	**0.00137**	**0.00141**
FPR	**0.00015**	**0.00036**	**0.0006**
mark	**8193.880**	**5037.838**	**6340.675**
Precision	0.9869	0.9828	0.9719
Recall	0.000775	0.00136	0.00141
F1_score	0.00154	0.00273	0.00282
ACC	0.06340	0.063947	0.06397
AUC_score	0.5003	0.5005	0.5004

Table 4 LOF Evolution Results in Test Data

Compare with the three features, the Feature mode 2 has a little better result that the TPR and FPR denotes a higher grade. However, the whole LOF model didn't fit well to detect the abnormal data in test dataset, because all three features' TPR and FPR scores contribute a really low mark.

Task II

1 Model Building (Logistic Regression)

First, I used the feature mode 3 (One-hot-PCA) as the features to extract data from A2 training dataset. Also make some pre-process of the data and set a label column in each dataset, where contains 'Botnet' word as 'I', not contains 'Botnet' key word as '0'. Then, I build a supervised model using Logical Regression algorithm, where set the parameters as auto.

2 Generate adversarial samples

I generate the adversarial samples by finding botnet traffic in the test dataset where labelled to '1' (botnet label).

Gradient detection is the weight parameter of the linear classifier. The sample was originally detected as 1, and then it was detected as 0 against the gradient.

Gradient descent is to use the negative gradient direction to determine the new search direction for each iteration so that during each iteration, the objective function to be optimized can be gradually reduced. In this model, I used the steepest descent law:

$$x(k+1) = x(k) - a * g(k) \Rightarrow botnet\_vec\_adversarial = botnet\_vec\_adversarial - model.coef\_ * rating$$

Among them, the 'rating' is the learning rate, or it can be a small constant. The trained logistic regression model is the gradient of adversarial samples.

The error can be measured by the error function J (epoch). The parameter epoch is the weight.

Adjust the weight epoch to make the minimum value of J(epoch). The gradient descent method is carried out in the following steps:

The first step of the gradient descent method is to give an initial value for epoch, here it is 1. Assuming that the initial value randomly is given is at the highest cross point: then adjust epoch in the direction of the gradient drop, which will make J (epoch) change in a lower direction. The end of the algorithm will be when the epoch falls to the point where it cannot continue to fall.

The overall implementation is as following:

```
botnet_vec=datas_map["test"][0][botnet_idx,:].reshape(1,-1)
botnet_label = datas_map["test"][1][botnet_idx]
botnet_vec_adversarial=botnet_vec.copy()
rating=0.001
epoch=0
while True:
    epoch=epoch+1
    res_prob = model.predict_proba(botnet_vec_adversarial)
    print("epoch:{},grad for sample,prob:{}".format(epoch, res_prob))
    label_pred =np.argmax(res_prob)
    if label_pred==1:
        botnet_vec_adversarial=botnet_vec_adversarial-model.coef_*rating
    else:
        print("success generate adversarial sample")
        break
```

Figure 18 Implement gradient descent-based method

The results is, after 372 times update in the gradient vector direction, it can not continue to fall, and the currently probabilities is [0.502, 0.497]. The adversarial samples generated successfully.

```
epoch:370,grad for sample,prob:[[0.49751403 0.50248597]]
epoch:371,grad for sample,prob:[[0.49996595 0.50003405]]
epoch:372,grad for sample,prob:[[0.50241786 0.49758214]]
success generate adversarial sample
```

Figure 19 Code results

3 Evolution

Type	Valid Results	Test Results
TN	0.215	0.101
FP	0.017	0.005
FN	0.537	0.602
TP	0.229	0.292
mark	**22.832**	**18.226**
Precision	0.928	0.983
Recall	0.299	0.326
F1_score	0.452	0.491
ACC	0.444	0.393
AUC_score	0.611	0.640

For our prediction results, the prediction accuracy of this model is very high, but the scores TP and TN of the samples that it correctly predicted to be positive are very low. So the model is good enough.

Conclusion and Discussion

Task I

In most anomaly detection scenarios, the data is unlabeled, and there are many more normal data than abnormal data in the data, so the unsupervised method is the most widely used method. The task I part mainly introduces unsupervised clustering techniques based on density and tree respectively. Most algorithms define similar results. However, some of them showed more stable output in terms of outlier consistency. The density-based OPTICS (DBSCAN-based) algorithm is a good attempt at the initial stage of training the model, but it is not suitable for high-dimensional data, and because it needs to traverse the data to calculate the distance, the computational complexity is high. It takes a long time, so this model is not suitable for this experiment. The tree-based IForest algorithm is very fast when tested, but it is not suitable for high-dimensional data, because high-dimensional data will have many irrelevant features. The LOF algorithm can better identify network abnormalities, that is, accurately identify attackers and victims, and the accuracy results are very high, but as the data dimension increases, the required computing resources increase greatly, and it is not suitable for online applications. In the end, I used the IForest algorithm for evaluation, and the final result was that the victim host was 145.116.120.45, and the host was attacked by 7 attackers: 145.178.112.177, 150.83.100.189, 144.181.96.193, 192.31.93.182, 145.74.96.192, 240.143.98.191, 145.74.96.192. In order to determine the performance of the model as an intrusion detection system (IDS), we also conducted a large number of model evaluations, such as calculating TP, FN, FP, TN, etc. to ensure the accuracy of the model.

Task II

First, for generating the adversarial samples in computer version, the inputs are more on images. Attacks will produce adversarial disturbances, so that the classifier will give false predictions to all output labels of the model, in order to deceive the semantic segmentation or object detection model. The attack found that for the semantic segmentation model, any input image has a general disturbance. As for network traffic, the inputs based on network flow.

Second, network traffic based on the time series, in terms of dataset.

Besides, there are many third libraries to support images model which are not fit the network aspects.

Reference

1. Flag State, [Online], [Access 19-10-2020]]https://www.cisco.com/c/en/us/support/docs/security/asa-5500-x-series-next-generation-firewalls/113602-ptn-113602.html
2. F. T. Liu, M. T. Kai, Z. H. Zhou. Isolation-Based Anomaly Detection[M]. ACM, 2012, 6 (1) :1-39

3. Xu W, Huang L, Fox A, et al. Detecting large-scale system problems by mining console logs[C]// ACM Sigops, Symposium on Operating Systems Principles. ACM, 2009:117-132.
4. Goldstein M, Uchida S. A Comparative Evaluation of Unsupervised Anomaly Detection Algorithms for Multivariate Data[J]. Plos One, 2016, 11(4):e0152173.
5. F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blon- del, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," Journal of Machine Learning Research, vol. 12, pp. 2825–2830, 2011.
6. F. T. Liu, K. M. Ting, and Z. Zhou, "Isolation Forest," in 2008 Eighth IEEE International Conference on Data Mining, Dec. 2008, pp. 413–422.
7. S. C. Tan, K. M. Ting, and T. F. Liu, "Fast Anomaly Detection for Streaming Data," in Twenty-Second International Joint Conference on Artificial Intelligence, Jun. 2011. [Online]. Available: https://www.aaai.org/ocs/index.php/IJCAI/IJCAI11/paper/ view/3229