COMP90073 Security Analytics, Semester 2 2020

Project 2: Machine learning based cyberattack detection

Name: Chen-An Fan
[redacted]

Task 1

1. Introduction

In this task 1, I use 3 different feature selection/generation methods and 2 different unsupervised clustering algorithms to detect the botnet traffic from the normal traffic. Totally, 6 models are generated. The botnet traffic is treated as anomaly. Our data sets contain 14 fields. The training set is used to train the models and the validation set is used to evaluate the performance and fine tune the parameters of models. Finally, I will use these 6 models to predict on the test set and generate CSV files that list the detected botnet attack.

2. Data Pre-processing

The given datasets are not clean. There are some pre-processing processes needed to be done before feeding them into the clustering algorithms. In this section, I will discuss the problems I need to address and method I pre-process them.

- **Missing Value**
  Missing Value is one of the most common problem in data. Inevitably, there are some missing values in fields *"Source Port", "Destination Port", "Source Service", "Destination Service"* and *"State"* of our data sets. These missing values need to be filled before further data pre-processing to prevent error from happening. I fill these missing values with value -1 or string "-1" depending on their data types.
- **Mixed Hexadecimal & Decimal Expressions in Port Fields**
  I found that, in the fields of "*source port"* and "*destination port"*, there are some port number are expressed in hexadecimal and some are expressed in decimal. To maintain the consistency of data, I convert all the port number into decimal expression to prevent confusion.
- **Removing Leading and Trailing Spaces**
  Some of the data in the *"direction"* field have leading or trailing spaces. In order to categorize the *"direction"* field correctly, they must be removed.
- **Encoding**
  Some of fields are categorial features, such as the *"state"* and *"direction"* fields. In order to feed them into Scilit-learn [1] clustering models, the categorial features must be encoded into numerical expressions.

3. Dataset Overview

In order to select the right features for training the clustering models to gain the best prediction result, we need to understand the dataset. In this section, I will examine the distribution and statistics of each fields in the test dataset by using Splunk and Python. (Please find the screenshots of data distribution in appendix a.)

There are total 1,053,845 instances in our test dataset with 14 fields, which are timestamp, duration, protocol,

source IP address, source port, direction, destination IP address, destination port, state, source type of service, destination type of service, the number of total packets, the number of bytes transferred in both directions, the number of bytes transferred from the source to the destination. Some of them are numerical features while others are categorial features. The distribution and statistics of them are show in table 1 and 2.

	Duration	*NoPackets*	*NoBytesBoth*	*NoBytesSrcToDst*
Mean	52.87	31.6	24148.21	19939.01
Std	274.05	4140.32	2131527	2014905
min	0	1	54	0
25%	0	2	197	115
50%	0.0014	3	383	220
75%	8.61	4	1071	1061
max	3925.12	3680561	945069500	983047100
Missing	0	0	0	0

Table 1. Distributions of Numerical Fields

	protocol	*srcIP*	*srcPort*	*direction*	*dstIP*	*dstPort*	*state*	*srcService*	*dstService*
Unique	6	325	91624	6	110765	27743	183	2	3
Top	TCP	224.134.91.164	2077	->	125.189.87.2	53	UNK	0	0
Top's Freq	420915	151166	46322	844228	92982	201888	331330	1053595	356150
Missing	0	0	72889	0	0	423743	4656	246	697648

Table 2. Distributions of Categorial Fields

Instances with extreme values are likely to be anomalies. As we can see in the table 1, these four fields (*duration, noPackets, noBytesBoth, noBytesSrcToDst*) have large standard deviation due to some extremely large values inside them. This is a good sing to use them for anomaly detection.

There are 6 types of *protocol* in our test data (the count of each type is shown in appendix a) most of them in the test dataset are TCP, ICMP, or UDP; only few of them are arp, rtcp, or rtp. This may be a helpful factor to detect anomalies.

On the other hand, the *srcIP, srcPort, dstIP, dstPort, state* fields are too trivial, directly using them may not help the anomalies detection. However, we can extract the interaction of these fields among instances. For example, we can count the number of the srcIP and dstIP showed in the whole dataset and then combine them with the ports. I believe the (IP count, port) combination could help the anomalies detection, because a single IP address showed many times in the dataset using the same port for communication is highly likely be a botnet server or botnet victim.

For the *srcService, dstService, direction* fields, the meaning of their value is not clear (for example, we do not know what does "*who*" mean in the *direction* fields and what does "*service 3*" mean). Nevertheless, their value distribution is very unbalanced; that is also a good sing for anomalies detection. Therefore, I also keep them to train my models.

4. Features Generating Methodology

In this section, I will discuss the methods I used to generate/select features for out clustering models.

a. Label-encoding and radius basis function (rbf) kernel

In the first method, I encode features by the normal encode technique (convert different feature values into continuous integers). I do not use one-hot encoding because the one-hot encoding would generate too many fields, which is unrealistic in this project. Then, I use the radius basis function kernel to project the input data into feature space. In this method, I dropped the *timestamp*, *srcIP* and *dstIP* fields, because it is not likely we can know the IP address of feature attackers or victims and the unique timestamp of each instance will not help to detect anomalies.

In order to maintain the extreme values in the data to facilitate anomalies detection, I do not standardize the training data in this method.

b. Generate new features to capture interaction between instances and normalize data

The previous features generating method has some disadvantage. First, it did not have features to describe the interaction between instances. Second, according to table 1, some features have much larger numerical value than others. These features with large numerical value may dominant the classifier.

To address the first issue, I generate two new features to capture the interaction between instances. These two features are the *dstIP_count* and *srcIP_count*, which represent the proportion of each instance's IP address in the whole dataset. As we can see in the table 3, some values of the *srcIP_count* are much larger than others. Which are likely botnet servers or victims. In this method, I will combine these two *IP_count* features with *port* features to facilitate the classification. As I have mentioned before, an (IP address, port) pair showed many times in the dataset is highly likely be a botnet server or victim.

	dstIP_count	*srcIP_count*
Mean	2.73%	3.56%
Std	2.88%	4.55%
min	9.5e-5%	3.8e-4%
25%	0.011%	0.99%
50%	2.03%	1.44%
75%	4.63%	3.77%
max	8.82%	14.3%
Missing	0	0

Table 3. Distributions of IP Count Features

To address the second issue, I normalize the whole data in every fields by the formula below.

$$z = \frac{x - \min(x)}{\max(x) - \min(x)}$$

Instead of using standardize which will decrease the impact of outliers, this normalization can map all the fields to [0,1] and maintain the outliers.

In this method, the *srcIP* and *dstIP* fields are dropped after I compute the *IP_count* fields. Also, the

timestamp fields are dropped because they are too trivial to use. I believe the new *IP_count* features and normalization may benefit the anomalies detection.

c. Linear Dimensionality Reduction by PCA

In the previous feature generation method, they are still too many fields and data. The time complexity of training a classifier is too high. Therefore, I apply PCA to the dataset generated by the method b to project them into lower dimensional space. This can speed the training process of my models, especially the OCSVM model.

5. Review of Two Anomaly Detection Methods

a. OCSVM

The first model I implemented is one-class support vector machine (OCSVM). OCSVM first using kernel to project the input data into higher dimensional feature space in order to separate the normal samples from the origin. Then, OCSVM iteratively finds the hyperplane with maximum margin to separate the training data and the origin. The data near the origin are treated as anomalies.

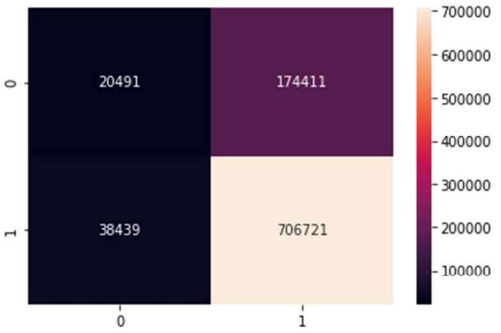
The standard support vector machine does not work well on the imbalance data and the number of anomalies is usually much smaller than the normal sample. Therefore, OCSVM which treated all data as one class and the origin as another data is suitable to address the imbalance issue of anomalies detection [2].

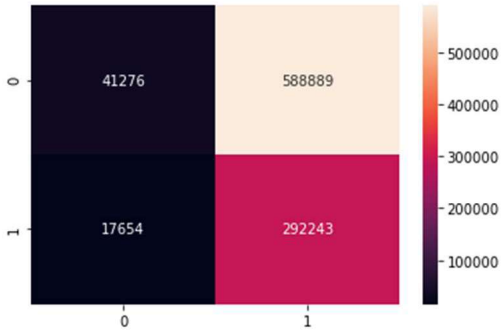
b. Isolation Forest

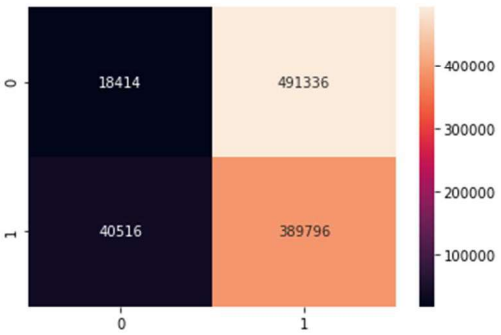
The second model I choose is isolation forest. Isolation forest are efficient to detect outlier in high-dimensional data which have a large number of unrelated features[3]. Isolation forest will first randomly pick a feature and then randomly select a split value between the maximum and minimum value of that selected feature to split the data into two set. This process will keep iterating to isolate the anomalies. Therefore, it is suitable to use a tree structure to represent the recursive partitioning. The anomalies are usually closer to the root node, while the normal samples usually have longer path. This algorithm has linear time complexity, therefore can easily scale up to our large dataset. Moreover, this algorithm converges quickly because it does not need to calculate distance or density based factors to detect the anomalies [3].

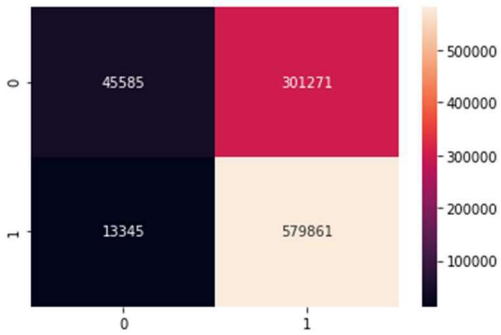
6. Experimental Setup & Evaluation

To evaluation the performance between three feature generation methods and two anomalies detection model. I set up experiments for all the combination evaluate them by predicting on the validation dataset. The anomalies detection result on the valid data of different model-feature combination are listed in table 4. The x-axis of confusion matrix showed in table 4 is the predicted classes and the y-axis is the true classes. Label 1 means anomalies and label 0 means normal sample. After I do the evaluation over the validation dataset, I feed the test dataset into these models to generate the attack log.

<table>
<tr><th></th><th>OCSVM (Sample 200000)</th><th>Isolation Forest</th></tr>
<tr><td>a. Label-encoding + rbf kernel</td><td>
<table>
<tr><th></th><th>Precision</th><th>Recall</th><th>F1-Score</th></tr>
<tr><td>Anomaly</td><td>0.80</td><td>0.95</td><td>0.87</td></tr>
<tr><td>Normal</td><td>0.35</td><td>0.11</td><td>0.16</td></tr>
<tr><td>Accuracy</td><td></td><td></td><td>0.77</td></tr>
<tr><td>Macro avg</td><td>0.57</td><td>0.53</td><td>0.52</td></tr>
<tr><td>Weighted avg</td><td>0.77</td><td>0.90</td><td>0.82</td></tr>
</table>


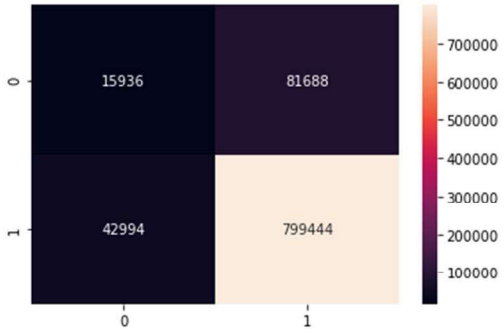
anomaly detected in test data: 712931
</td><td>
<table>
<tr><th></th><th>Precision</th><th>Recall</th><th>F1-Score</th></tr>
<tr><td>Anomaly</td><td>0.33</td><td>0.94</td><td>0.49</td></tr>
<tr><td>Normal</td><td>0.70</td><td>0.07</td><td>0.12</td></tr>
<tr><td>Accuracy</td><td></td><td></td><td>0.35</td></tr>
<tr><td>Macro avg</td><td>0.52</td><td>0.5</td><td>0.31</td></tr>
<tr><td>Weighted avg</td><td>0.35</td><td>0.89</td><td>0.47</td></tr>
</table>


anomaly detected in test data: 496167
</td></tr>
<tr><td>b. New features + normalize</td><td>
<table>
<tr><th></th><th>Precision</th><th>Recall</th><th>F1-Score</th></tr>
<tr><td>Anomaly</td><td>0.44</td><td>0.91</td><td>0.59</td></tr>
<tr><td>Normal</td><td>0.31</td><td>0.04</td><td>0.06</td></tr>
<tr><td>Accuracy</td><td></td><td></td><td>0.43</td></tr>
<tr><td>Macro avg</td><td>0.38</td><td>0.47</td><td>0.33</td></tr>
<tr><td>Weighted avg</td><td>0.43</td><td>0.58</td><td>0.56</td></tr>
</table>


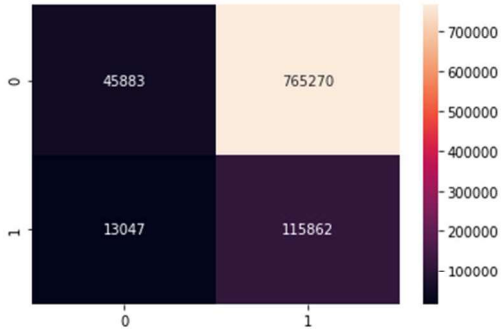
anomaly detected in test data: 426197
</td><td>
<table>
<tr><th></th><th>Precision</th><th>Recall</th><th>F1-Score</th></tr>
<tr><td>Anomaly</td><td>0.66</td><td>0.98</td><td>0.79</td></tr>
<tr><td>Normal</td><td>0.77</td><td>0.13</td><td>0.22</td></tr>
<tr><td>Accuracy</td><td></td><td></td><td>0.76</td></tr>
<tr><td>Macro avg</td><td>0.72</td><td>0.55</td><td>0.51</td></tr>
<tr><td>Weighted avg</td><td>0.67</td><td>0.92</td><td>0.75</td></tr>
</table>


anomaly detected in test data: 451183
</td></tr>
<tr><td>c. New features + PCA</td><td>
<table>
<tr><th></th><th>Precision</th><th>Recall</th><th>F1-Score</th></tr>
<tr><td>Anomaly</td><td>0.91</td><td>0.95</td><td>0.93</td></tr>
</table>
</td><td>
<table>
<tr><th></th><th>Precision</th><th>Recall</th><th>F1-Score</th></tr>
<tr><td>Anomaly</td><td>0.13</td><td>0.90</td><td>0.23</td></tr>
</table>
</td></tr>
</table>

Normal	0.27	0.16	0.20	Normal	0.78	0.06	0.11
Accuracy			0.87	Accuracy			0.17
Macro avg	0.59	0.56	0.57	Macro avg	0.46	0.48	0.17
Weighted avg	0.87	0.90	0.88	Weighted avg	0.17	0.85	0.22



anomaly detected in test data: 823238



anomaly detected in test data: 205727

Table 4. Model Evaluation

We use precision and recall to measure the model performance.

$$Precision = \frac{TP}{TP + FP} = \frac{\#\ Correctly\ detected\ anomalies}{\#\ Total\ detected\ anomalies}$$

$$Recall\ (True\ Positive\ Rate) = \frac{TP}{TP + FN} = \frac{\#\ Correctly\ detected\ anomalies}{\#\ Actual\ anomalies}$$

Precision indicates how many detected anomalies are actually anomalies.

Recall indicates how many actual anomalies are detected correctly.

7. Discussion

As we can see in the table 4, the precision of normal data is higher than anomalies while the recall of anomalies is higher than normal data in isolation forest models. This means that the anomalies are more likely be found than the normal data, however, we have more confidence when we predict data as normal by using isolation forest. This may be caused by the nature of isolation tree; the anomalies easier to be found on the node near the root but we are not sure how near they should be. On the other hand, the normal data are found in the very bottom of the tree, therefore, when we see a data with long path from root, we have high confidence that it should be a normal sample.

We can also find that the precision and recall of anomalies are both higher than the normal data in OCSVM models. This may be caused by the data distribution difference between training set and validation set. Most of instances in our training set are normal sample, however, over 90% of instances in the validation set are anomalies. Therefore, the precision and recall of normal samples are much worse than the anomalies.

Overall, the best combination is using OCSVM with feature generation method c. The reason OCSVM outperformance than isolation forest is that the features in our dataset are not fully irrelevant. The isolation forest models ignore the interaction between features, such as the interaction between IP and port. Furthermore, I found that generating features by PCA lead to the best performance for the OCSVM model.

By using PCA to reduce the dimension of features, I could sample more training data to train the OCSVM, therefore, I get a better result.

8. CSV File

I use the six models to predict on the test data set. The detected botnet attacks are saved in csv files. Each instance represents an attack scenario with timestamp, IP address, and protocol.

Task 2

1. Introduction

In this task, I use multilayer perceptron (MLP) to train a supervised machine learning model for anomalies detection. The data is pre-processed and encoded by the same method described in task 1. And the features are selected by the feature generation method b in task 1. Then I would generate adversarial samples for a chosen botnet IP to bypass the detection of my MLP model.

2. Building MLP Model

The MLP model is training on the training set and using the test set to monitor the loss. After training, the validation set is fed into the model for evaluation.

The best configuration of the MLP model I found is:

hidden_layer_sizes=(8,6,4), activation = 'logistic', solver='adam', learning_rate_init=0.001, alpha=0.0001

	Precision	Recall	F1-Score
Anomaly	0.99	0.97	0.98
Normal	0.89	0.95	0.92
Accuracy			0.96
Macro avg	0.94	0.96	0.95
Weighted avg	0.96	0.96	0.96

Table 5. MLP Model Evaluation on Valid Set

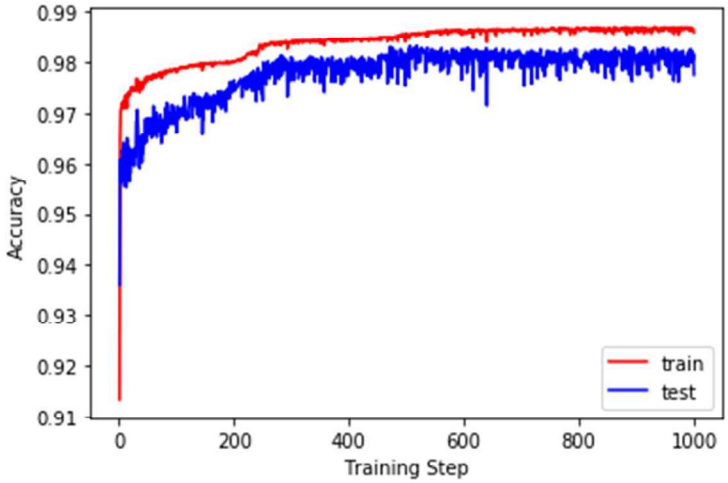


Figure 1. Training Curve of MLP Model

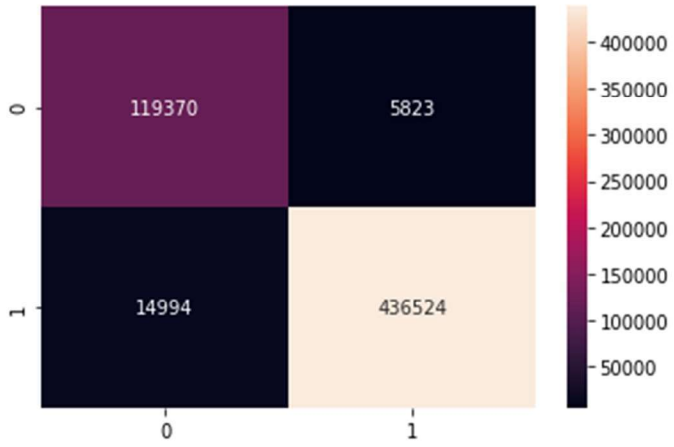


Figure 2. Confusion Matrix of MLP Model (X: predicted, Y: true | 1: anomalies, 0: normal)

As we can see in the table 5 and figure 2, the precision and recall of both normal and anomaly is very high. The MLP model converge fast and reach 90% of testing accuracy within 100 steps as figure 1 indicated.

3. Generating Adversarial Example

To generate adversarial example (change the dataset without change the model), I want to use Tensorflow FGSM but I failed to do that.

The FGSM is a white box attack. It modifies the data to make sure the model misclassification. It can bypass the model detection without changing the model.

The fast gradient sign method uses the gradient of the neural network to generate adversarial example. The goal is to maximize the loss with minimal effort. This is achieved by measuring how much each feature in the dataset contributes to the loss and then add a perturbation accordingly [4].

4. Discussion

- How to update its network traffic in order to satisfy the modified features
  The network traffic is captured online, therefore, we could not modify it after it is captured by other people. We can only modify the botnet traffic before the package is sent. For example, change port every time we launch a botnet attack and split a long attack into multiple short commands with different size. These can avoid the attack pattern be detected and therefore bypass the detection model.
- Main difference between generating adversarial samples in computer vision and in network traffic
  In computer vision, we can manipulate the image freely. For example, add distortion to the images. However, the network traffic is real-time; it's not possible to manipulate the data after the traffic is sent. Therefore, all the adversarial samples generation need to be done before be sent and they should maintain the same malicious functionality to achieve bypass and attack at the same time.

Reference

[1] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *the Journal of machine Learning research,* vol. 12, pp. 2825-2830, 2011.

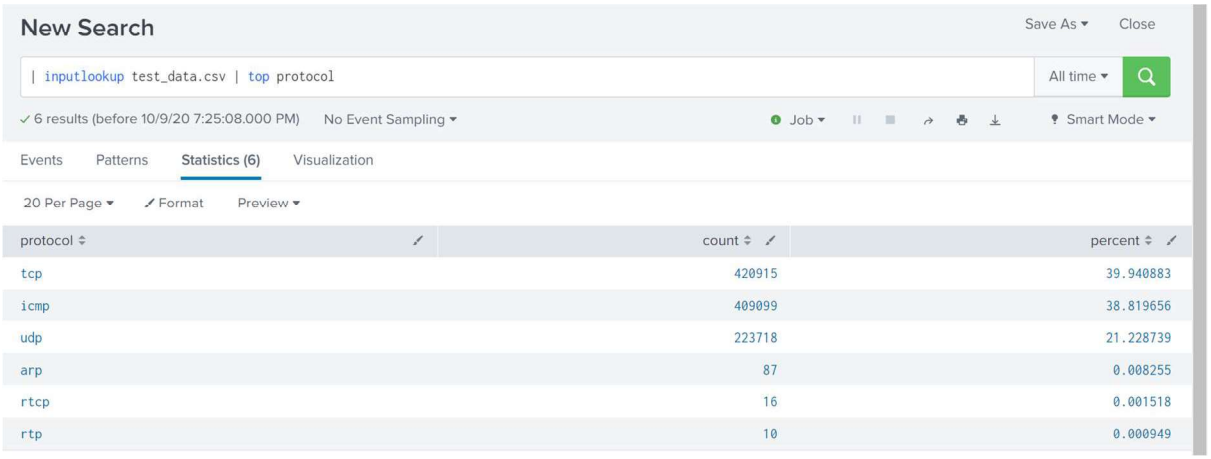
[2] B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson, "Estimating the support of a high-dimensional distribution," *Neural computation,* vol. 13, no. 7, pp. 1443-1471, 2001.

[3] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *2008 Eighth IEEE International Conference*

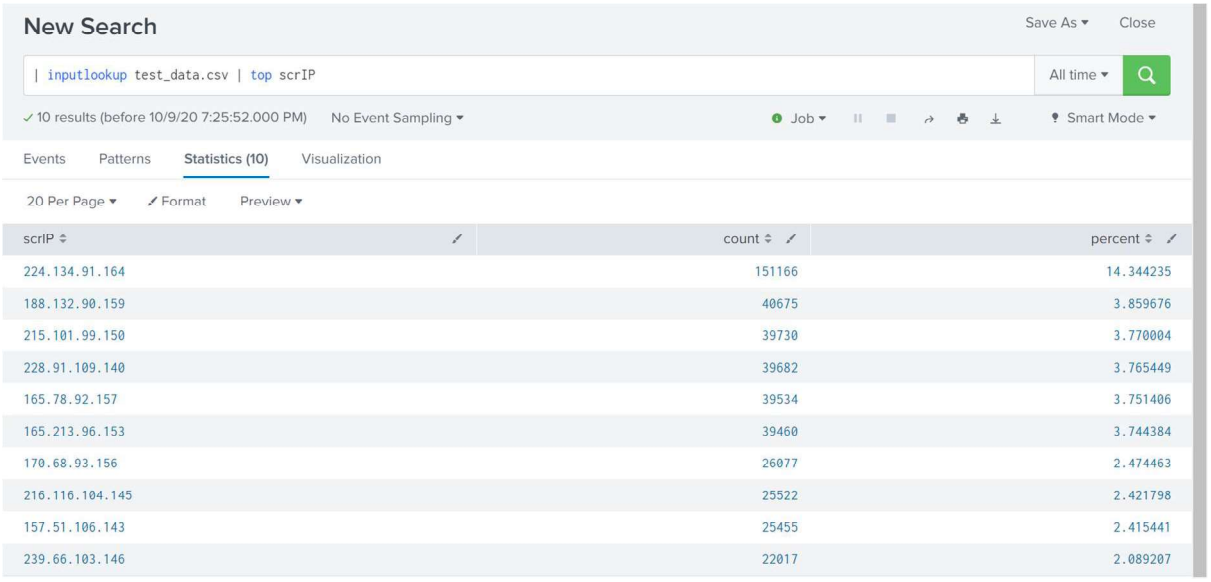
on Data Mining, 2008: IEEE, pp. 413-422.

[4] TendorFlow. "Adversarial example using FGSM." https://www.tensorflow.org/tutorials/generative/adversarial_fgsm (accessed 20,11, 2020).

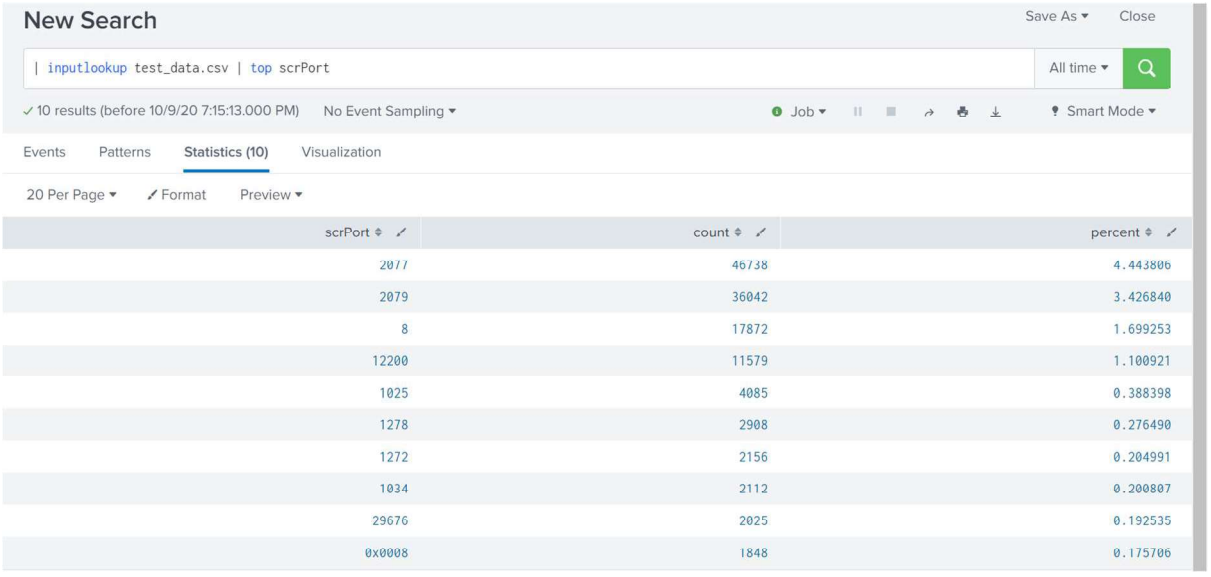
Appendix A – Data Distribution of Fields



protocol	count	percent
tcp	420915	39.940883
icmp	409099	38.819656
udp	223718	21.228739
arp	87	0.008255
rtcp	16	0.001518
rtp	10	0.000949



scrIP	count	percent
224.134.91.164	151166	14.344235
188.132.90.159	40675	3.859676
215.101.99.150	39730	3.770004
228.91.109.140	39682	3.765449
165.78.92.157	39534	3.751406
165.213.96.153	39460	3.744384
170.68.93.156	26077	2.474463
216.116.104.145	25522	2.421798
157.51.106.143	25455	2.415441
239.66.103.146	22017	2.089207



scrPort	count	percent
2077	46738	4.443806
2079	36042	3.426840
8	17872	1.699253
12200	11579	1.100921
1025	4085	0.388398
1278	2908	0.276490
1272	2156	0.204991
1034	2112	0.200807
29676	2025	0.192535
0x0008	1848	0.175706

New Search

Save As ▾ Close

```
| inputlookup test_data.csv | top direction
```

All time ▾

✓ 6 results (before 10/9/20 7:27:25.000 PM) No Event Sampling ▾ Job ▾ Smart Mode ▾

Events Patterns **Statistics (6)** Visualization

20 Per Page ▾ Format Preview ▾

direction	count	percent
->	844228	80.109314
<->	207315	19.672248
?>	1695	0.160840
<?>	516	0.048964
who	87	0.008255
<-	4	0.000380

New Search

Save As ▾ Close

```
| inputlookup test_data.csv | top destIP
```

All time ▾

✓ 10 results (before 10/9/20 7:33:12.000 PM) No Event Sampling ▾ Job ▾ Smart Mode ▾

Events Patterns **Statistics (10)** Visualization

20 Per Page ▾ Format Preview ▾

destIP	count	percent
125.189.87.2	92982	8.823119
160.108.108.57	67620	6.416503
202.18.114.51	61332	5.819831
205.224.112.53	48834	4.633888
175.225.124.41	46975	4.457487
169.17.118.47	45902	4.355669
141.111.103.62	36807	3.492639
204.213.241.7	33290	3.158909
195.231.110.55	32827	3.114974
145.116.120.45	25541	2.423601

New Search

Save As ▾ Close

```
| inputlookup test_data.csv | top destPort
```

All time ▾

✓ 10 results (before 10/9/20 7:33:56.000 PM) No Event Sampling ▾ Job ▾ Smart Mode ▾

Events Patterns **Statistics (10)** Visualization

20 Per Page ▾ Format Preview ▾

destPort	count	percent
53	203539	30.679754
25	80642	12.155296
22	77200	11.636478
443	59836	9.019175
80	55125	8.309078
135	21312	3.212391
6667	8674	1.307446
13363	8010	1.207360
3389	4922	0.741901
179	4281	0.645282

New Search

Save As ▾ Close

```
| inputlookup test_data.csv | top state
```

All time ▾

✓ 10 results (before 10/9/20 7:40:53.000 PM) No Event Sampling ▾ Job ▾ Smart Mode ▾

Events Patterns **Statistics (10)** Visualization

20 Per Page ▾ Format Preview ▾

state	count	percent
UNK	331330	31.579630
S_	269510	25.687460
CON	207237	19.752113
S_RA	71505	6.815264
FSPA_FSPA	40407	3.851260
ECO	19720	1.879547
INT	16472	1.569975
ROB	16072	1.531850
SRPA_SPA	7826	0.745909
FSPA_FSA	5872	0.559670

New Search

Save As ▾ Close

```
| inputlookup test_data.csv | top srcService
```

All time ▾

✓ 2 results (before 10/9/20 7:51:27.000 PM) No Event Sampling ▾ Job ▾ Smart Mode ▾

Events Patterns **Statistics (2)** Visualization

20 Per Page ▾ Format Preview ▾

srcService	count	percent
0	1053595	99.999620
3	4	0.000380

New Search

Save As ▾ Close

```
| inputlookup test_data.csv | top dstService
```

All time ▾

✓ 3 results (before 10/9/20 7:52:16.000 PM) No Event Sampling ▾ Job ▾ Smart Mode ▾

Events Patterns **Statistics (3)** Visualization

20 Per Page ▾ Format Preview ▾

dstService	count	percent
0	356150	99.986805
2	29	0.008142
3	18	0.005053

New Search

Save As ▾ Close

```
| inputlookup test_data.csv | top NoPackets
```

All time ▾

✓ 10 results (before 10/10/20 12:56:20.000 AM) No Event Sampling ▾ Job ▾ Smart Mode ▾

Events Patterns **Statistics (10)** Visualization

20 Per Page ▾ Format Preview ▾

NoPackets	count	percent
2	301464	28.606104
3	284646	27.010234
4	142046	13.478832
1	118453	11.240078
5	42222	4.006472
7	34046	3.230646
8	21351	2.026010
6	18496	1.755097
11	10758	1.020833
10	9421	0.893964

New Search

Save As ▾ Close

```
| inputlookup test_data.csv | top NoBytesBoth
```

All time ▾

✓ 10 results (before 10/10/20 12:57:17.000 AM) No Event Sampling ▾ Job ▾ Smart Mode ▾

Events Patterns **Statistics (10)** Visualization

20 Per Page ▾ Format Preview ▾

NoBytesBoth	count	percent
185	8754	0.830672
184	8677	0.823366
186	8444	0.801256
183	8184	0.776585
187	7918	0.751344
182	7572	0.718512
188	7316	0.694220
181	7181	0.681410
61	7098	0.673534
189	7087	0.672490

New Search

Save As ▾ Close

```
| inputlookup test_data.csv | top NoBytesSrcToDst
```

All time ▾

✓ 10 results (before 10/10/20 12:58:18.000 AM) No Event Sampling ▾ Job ▾ Smart Mode ▾

Events Patterns **Statistics (10)** Visualization

20 Per Page ▾ Format Preview ▾

NoBytesSrcToDst	count	percent
72	11878	1.127111
71	11630	1.103578
184	11458	1.087257
185	11365	1.078432
73	11309	1.073118
186	10919	1.036111
183	10907	1.034972
74	10852	1.029753
70	10676	1.013052
187	10142	0.962381

New Search

Save As ▾ Close

```
| inputlookup test_data.csv | top duration
```

All time ▾

✓ 10 results (before 10/10/20 2:34:08.000 AM) No Event Sampling ▾ Job ▾ Smart Mode ▾

Events Patterns **Statistics (10)** Visualization

20 Per Page ▾ Format Preview ▾

duration	count	percent
0.0	424873	40.316460
0.000303	503	0.047730
0.000309	463	0.043934
0.000302	462	0.043839
0.000308	461	0.043745
0.000299	459	0.043555
0.000266	458	0.043460
0.000258	444	0.042131
0.000261	443	0.042037
0.000316	442	0.041942