COMP90073
Project 2: Machine learning based cyberattack detection

Hongsang Yoo

Student ID: 1105791

1. Overview

1.1 Dataset

The provided datasets consist of 3 parts: training, validation and test data. Training and test dataset for task 1 do not contain labels since the purpose of the task is unsupervised anomy detection. However, for the purpose of tuning hyperparameters, validation dataset was provided with labels. For task 2, all datasets have labels. These datasets were first loaded in Splunk for overview and inspection of data and later used Python for generating features and applying them to models.

1.2 Summary of attack

IP of attacker	IP of victim	protocol	Start time	End time
224.134.91.164	125.189.87.2	Icmp	2012-12-02 15:03:04	2012-12-08 04:56:28
224.134.91.164	125.189.87.2	udp	2012-12-01 15:44:12	2012-12-10 15:09:29
224.134.91.164	204.213.241.7	tcp	2012-12-01 15:56:34	2012-12-08 11:22:03
215.101.99.150	162.52.232.25	tcp	2012-12-05 01:24:32	2012-12-06 01:24:32
228.91.109.140	135.31.242.15	tcp	2012-12-09 23:24:33	2012-12-10 23:24:32

Table 1: Attack overview

2. Feature generation

2.1 Features used in project1

Previously, in the project 1, since the query was asked through Splunk and the process of narrowing scope kept going on, the interesting fields were not strictly set at first. At the end of the project, some important features were extracted as seen in Table2. Generally, combination of src_ip, src_port, dst_ip, dst_port with other fields were the most frequently used.

Feature	Description
start	Start of the conversation
finish	End of the conversation
src_ip	Source IP
dst_ip	Destination IP
src_port	Source port
dst_port	Destination port
num_src_ip	Number of source IPs
num_src_port	Number of source ports
num_dst_ip	Number of destination IPs
num_dst_port	Number of destination ports
info	Information sent via packet

Table2: Features used in project1

2.2 Generation of new features

The provided dataset had 14 columns which contain categorical and numeric values. Based on these existing columns, some were modified/processed, removed and added. Contrast to Project 1, categorical fields such as these (src_ip, src_port, dst_ip, dst_port) were not used explicitly in some feature sets, but still showed good performance.

In this project, 3 types of different methods for feature generation were used. For all feature sets, PCA and Standardscaler were applied. PCA was used for dimensionality reduction so the training process gets faster and generalises models more preventing overfitting.

Regarding feature selection, at first only numeric fields were chosen and generating other fields based on them (pps, bps, bpp). And then Feature 2 incorporated one-hot encoded categorical feature into this. Feature 1 and feature 2 are based on data from each row in dataset so generates the same number of lines as the original dataset. However, feature 3 was generated grouped by 5 columns so the new dataset contains fewer rows than the original dataset.

2.2 Feature 1: Numeric value (existing + newly generated) + Standardscaler + PCA

The features were overviewed and generated via query in Splunk first as seen below. Features selected are as seen in query in Figure 1 and Table 2.

```
source="c:\\program files\\splunk\\etc\\apps\\splunkforpcap\\pcapcsv\\training_data.csv" protocol="icmp"
| eval flow_id = src_ip + ":" + src_port + direction +  dst_ip + ":" + dst_port
| eval pps = total_packets/duration
| eval bps_onedir = bytes_onedir/duration
| eval bpp_onedir = bytes_onedir/total_packets
| eval bps_bothdir = bytes_bothdir/duration
| eval bpp_bothdir = bytes_bothdir/total_packets
| fillnull timestamp, duration, protocol, src_ip, src_port, direction, dst_ip, dst_port,
state, src_type, dst_type, total_packets, bytes_bothdir, bytes_onedir
| table src_ip, src_port, direction, dst_ip, dst_port, timestamp, duration,protocol, total_packets, bytes_onedir, bytes_bothdir, pps, bps_onedir,
    bpp_onedir, bps_bothdir, bpp_bothdir, state, src_type, dst_type
```

All time

Figure 1: SPL

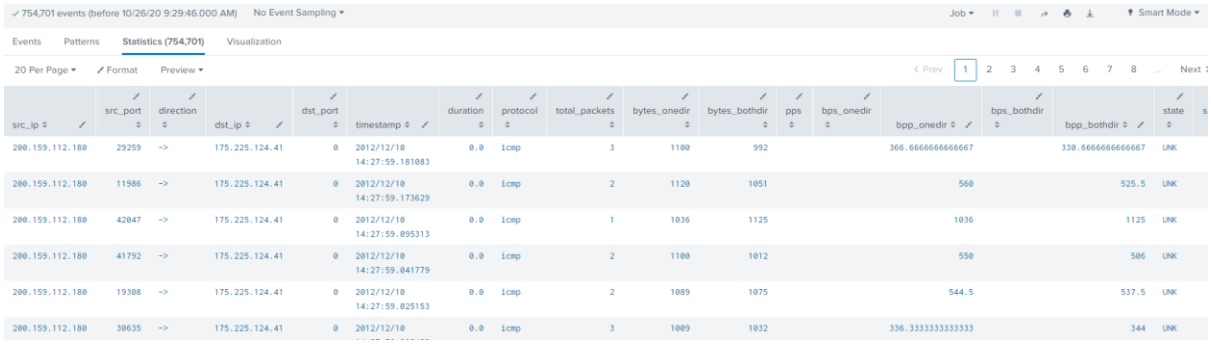


Figure 2: Search result

However, due to the lack of expertise in SPL, some fields such as pps, bps_onedir, bps_bothdir were missing because they were divided by duration, which is zero value. In addition, there were many fields that need to be preprocessed more in detail so these fields were extracted in Python later.

Feature	Description
total_packets	The number of total packets
bytes_onedir	the number of bytes transferred from the source to the destination
bytes_bothdir	the number of bytes transferred in both directions
duration	Duration of the conversation
pps	Number of Packets per second
bps_onedir	Number of bytes per second transferred in one direction
bpp_bothdr	Number of bytes per packet transferred in both directions
bps_onedir	Number of bytes per second transferred in one direction
bpp_bothdir	Number of bytes per packet transferred in both directions

Table3: Feature1 (numeric values + PCA)

2.3 Feature 2: Feature1 + One-hot encoded categorical feature

In addition to feature 1, categorical features (src_ip, dst_ip, src_port, dst_port, protocol) were added. Only top 10 occurring encoded features were used since I expected a memory problem if all categorical features were one-hot encoded. More detailed analysis is done is section 3 but Feature 2 showed a better performance than Feature 1.

```
colvalues_count = df[column].value_counts()
len_ccolvalues_count = 0

if column == 'src_ip':
    len_ccolvalues_count = 10
elif column == 'src_port':
    len_ccolvalues_count = 10
```

Figure 3: one-hot encoding threshold

Feature	Description
total_packets	The number of total packets
bytes_onedir	the number of bytes transferred from the source to the destination
bytes_bothdir	the number of bytes transferred in both directions
duration	Duration of the conversation
pps	Number of Packets per second
bps_onedir	Number of bytes per second transferred in one direction
bpp_bothdr	Number of bytes per packet transferred in both directions
bps_onedir	Number of bytes per second transferred in one direction
bpp_bothdir	Number of bytes per packet transferred in both directions
src_ip	Top 10 one hot encoded source IP
dst_ip	Top 10 one hot encoded destination IP
src_port	Top 10 one hot encoded source port
dst_port	Top 10 one hot encoded destination IP
protocol	Top 10 one hot encoded protocol

Table 4: Feature2 (Feature1 + One hot encoded categorical feature)

2.4 Feature 3: Scale(Cumulative features grouped by stream_id + time-based feature) + PCA

The last feature was generated grouped by (src_ip, dst_ip, src_port, dst_port, protocol) thus it contains unique rows based on these columns. The idea of grouping was got from the lecturer's comment in the discussion board. Basically, Feature 1 was summed/averaged grouped by the 5 columns above. And time- based features (cnt_timed_src, cnt_timed_dst, cnt_timed_srcdst) were added because anomaly traffic may be concentrated in a very short period of time.

Feature	Description
total_packets	Sum of the number of total packets in one conversation grouped by stream_id
bytes_onedir	Sum of the number of bytes transferred from the source to the destination in one conversation grouped by stream_id
bytes_bothdir	Sum of the number of bytes transferred in both directions in one conversation grouped by stream_id
duration	Sum of Duration of the conversation
pps	Grouped sum of the number of Packets per second
bps_onedir	Grouped sum of the number of number of bytes per second transferred in one direction
bpp_bothdr	Grouped sum of the number of number of bytes per packet transferred in both directions
bps_onedir	Grouped sum of the number of number of bytes per second transferred in one direction
bpp_bothdir	Grouped sum of the number of number of bytes per packet transferred in both directions
cnt_timed_src	Count of transmission sent by one src_ip within 1-minute time frame.
cnt_timed_dst	Count of transmission sent to one dst_ip within 1-minute time frame.
cnt_timed_srcdst	Count of transmission sent by one src_ip to one dst_port within 1-minute time frame.

Table 5: Feature3 (Grouped cumulative features + time-based feature + PCA)

3. Anomaly detection

3.1 Models

For models, Iforest and OneclassSVM was used. The criteria of choosing these models included the fact that they are efficient and can separate training process from testing to predict the result of test data. The IForest algorithm separates observations by random selection of features and split value between the maximum and minimum values. Rather than profiling normal data points, this method separates anomalies from them [1].

OCSVM (OneclassSVM) defines a decision boundary based on the distribution of the dataset so that it predicts data points to one class similar to the distribution of training dataset. Otherwise, it predicts as the other class. By its nature, it is useful especially for imbalanced dataset because what the model is interested in is the distribution of the dataset.

3.2 Hyperparameter tuning

The threshold in setting hyperparameters in my experiment was as following:

Accuracy > 0.88 and Max(TPR-FPR)

```
if accuracy> 0.88 and TPR_FPR_DIFF > best_score:
    best_acc = accuracy
    best_score = TPR_FPR_DIFF
    best_param = param_dict
```

Since the validation set is very largely biased towards anomaly, if one says that all the data are anomaly, they can get easily 90% accuracy easily. Generally, the models showed a good performance towards majority data (anomaly in case of the validation set). However, the precision and recall for the minority data set (normal data in case of the validation set) was very low. Therefore, while trying to meet the minimum accuracy, I tried to maximise the difference between TPR and FPR.

The process of hyperparameter tuning was performed against validation set. Popular methods such as Gridsearch was not used because it could not separate the training process from validation dataset evaluating only using validation set unless customized. Instead, I implemented it such that it should be trained using train dataset only and evaluated against validation set using a different combination of hyperparameters.

For OCSVM model, there are some important hyperparameters in this model: kernel , nu, gamma. For kernel, “rbf” always showed the best performance regardless of other hyperparameters so did not include other parameters in the tuning process. “Nu” value is related to accepting the scope of anomalies. Gamma is used as coefficient for Kernels [2] and was chosen according to the best scores set above.

For Iforest, while most of the parameters were the best by default through experiment, some meaningful parameters worth tuning included: max_samples and contamination. The parameter “max_samples” means the number of samples to draw to train each base estimator and the parameter “contamination” means the proportion of outliers) [3].

The hyperparameter tuning result for each model and feature is as follows:

Here, “best score” means the best (TPR-FPR) and “best ACC” is the accuracy when “best score” is chosen while meeting the least accuracy as a threshold (0.88)

1) Feature1 + OSCVM

```
now... >>  25 / 25
index:  25 , kernel:  rbf , gamma:  0.2 , nu:  0.2
fit:  11.190074682235718
predict:  5.050496816635132
46368 3086 96 1213
TPR_FPR_DIFF:  0.004676335386316843 accuracy:  0.9153123337864193

==========================
best score:  0.20954418381669737  || best ACC:  0.9066840021275339  || best param:  {'kernel': 'rbf', 'gamma': 0.1, 'nu': 0.1}
```

Best parameter: {'kernel': 'rbf', 'gamma': 0.1, 'nu': 0.1}

>> Without PCA (same feature and model as above)

```
now... >>  25 / 25
index:  25 , kernel:  rbf , gamma:  0.2 , nu:  0.2
fit:  14.246896982192993
predict:  6.42484974861145
47580 3175 7 1
TPR_FPR_DIFF:  0.0021788575004804134 accuracy:  0.9374347457794062

==========================
best score:  0.15718680312936473  || best ACC:  0.9326477946535863  || best param:  {'kernel': 'rbf', 'gamma': 0.1, 'nu': 0.1}
```

As seen above pictures, accuracy did not change much whether PCA was applied. However, when PCA was applied, there was a good increase in TPR-FPR score (best score in the above picture). It means that the model is more generalised and less susceptible to overfitting.

2) OCSVM + Feature 2

```
now... >>  25 / 25
index:  25 , kernel:  rbf , gamma:  0.2 , nu:  0.2
fit:  11.637219190597534
predict:  5.119346857070923
46962 2792 390 619
TPR_FPR_DIFF:  0.10955503038379588 accuracy:  0.9328053897523787

==========================
best score:  0.32835869643629045  || best ACC:  0.9259697023422572  || best param:  {'kernel': 'rbf', 'gamma': 0.01, 'nu': 0.1
5}
```

Best parameter: {'kernel': 'rbf', 'gamma': 0.01, 'nu': 0.15}

3) OCSVM + Feature 3

```
now... >>  25 / 25
index:  25 , kernel:  rbf , gamma:  0.2 , nu:  0.2
fit:  11.637219190597534
predict:  5.119346857070923
46962 2792 390 619
TPR_FPR_DIFF:  0.10955503038379588 accuracy:  0.9328053897523787

==========================
best score:  0.32835869643629045  || best ACC:  0.9259697023422572  || best param:  {'kernel': 'rbf', 'gamma': 0.01, 'nu': 0.1
5}
```

{'kernel': 'rbf', 'gamma': 0.15, 'nu': 0.2}

4) Iforest + Feature 1

```
now... >>  20 / 20
index:  20 , max_samples:  0.8 , contamination:  0.5
47581 3182 0 0
TPR_FPR_DIFF:  0.0 accuracy:  0.937316549455312

==========================
best score:  0.0021998742928975856  || best ACC:  0.9374544451667554  || best param:  {'max_samples': 0.4, 'contamination': 0.
1}
```

Best parameter: {'max_samples': 0.4, 'contamination': 0.1}

5) Iforest + Feature 2

```
now... >>  20 / 20
index:  20 , max_samples:  0.8 , contamination:  0.5
47581 3182 0 0
TPR_FPR_DIFF:  0.0 accuracy:  0.937316549455312

==========================
best score:  0.040565559902475234  || best ACC:  0.9233890825995311  || best param:  {'max_samples': 0.4, 'contamination': 0.1}
```

Best parameter: {'max_samples': 0.4, 'contamination': 0.1}

6) Iforest + F3

```
now... >>  20 / 20
index:  20 , max_samples:  0.8 , contamination:  0.5
47581 3182 0 0
TPR_FPR_DIFF:  0.0 accuracy:  0.937316549455312

==========================
best score:  0.019177922154144378  || best ACC:  0.9289049110572661  || best param:  {'max_samples': 'auto', 'contamination':
0.3}
```

Best parameter: {'max_samples': 'auto', 'contamination': 0.3}

4. Analysis

4.1 Clustering visualisation and Evaluation

Generally, Iforest was fast and showed a good accuracy but was weak at generalisation. In other words, the score for the difference between (TPR – FPR) was low. In contrast, OCSVM showed better performance in the score for (TPR – FPR). The classification reports and the clustering visualisation is seen below. Note that in clustering pictures, data classified as “Normal” are centred around zero in X-axis, while data classified as “Anomaly” are scattered around outside the scope of “Normal data”.

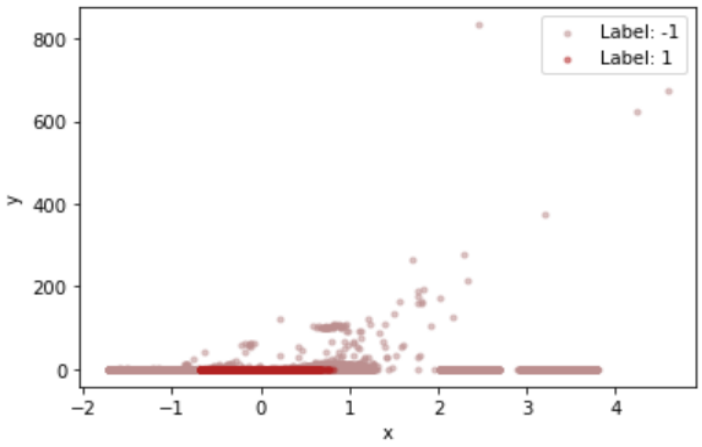
1) OCSVM + Feature 1

>> SCORES:

```
========== OCSVM: FEATURE 1 ==========
              precision    recall  f1-score   support

          -1       0.95      0.95      0.95     47581
           1       0.26      0.26      0.26      3182

    accuracy                           0.91     50763
   macro avg       0.60      0.60      0.60     50763
weighted avg       0.91      0.91      0.91     50763

| TP:  45200 | FP:  2356 | TN:  826 | FN:  2381
TPR:  0.9499590172547866
FPR:  0.7404148334380892
TPR - FPR :  0.20954418381669737
```

>> CLUSTERING:

```
=======>>>OCSVM CLUSTER RESULT

Exporting LABEL...
Number of clusters: 2
\Label -1: 1035114 data points
\Label 1: 18731 data points

DONE.
--------------------------------------
Wall time: 1min
```



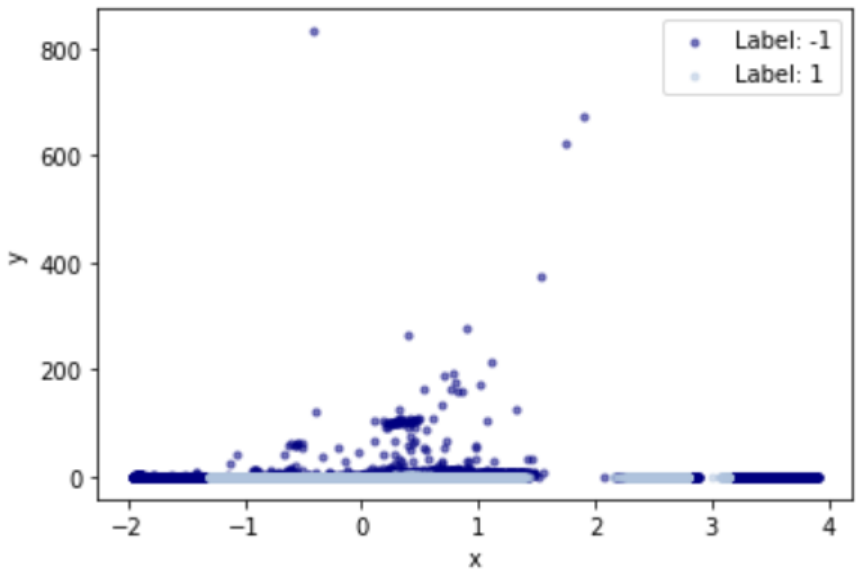
2) OCSVM + Feature 2

>> SCORES:

```
========== OCSVM: FEATURE 2 ==========
              precision    recall  f1-score   support

          -1       0.96      0.96      0.96     47581
           1       0.40      0.36      0.38      3182

    accuracy                           0.93     50763
   macro avg       0.68      0.66      0.67     50763
weighted avg       0.92      0.93      0.92     50763

| TP:  45844 | FP:  2021 | TN:  1161 | FN:  1737
TPR:  0.9634938315714255
FPR:  0.6351351351351351
TPR - FPR :  0.32835869643629045
```

>> CLUSTERING:

```
========== OCSVM FEATURE 2 ==========
=======>>>OCSVM CLUSTER RESULT

Exporting LABEL...
Number of clusters: 2
\Label -1: 850374 data points
\Label 1: 203471 data points

DONE.
---------------------------------------
Wall time: 1min 32s
```



3) OCSVM + Feature 3

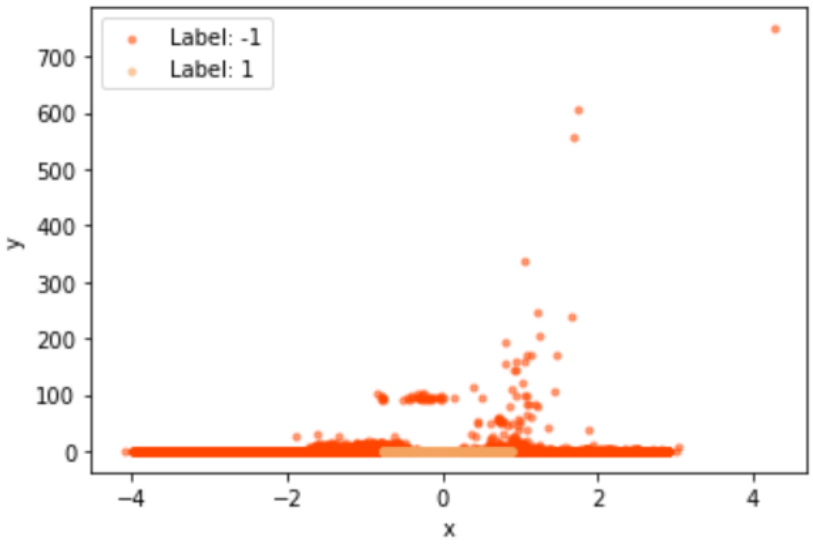
>> SCORES:

```
========== OCSVM: FEATURE 3 ==========
              precision    recall  f1-score   support

          -1       0.96      0.96      0.96     41267
           1       0.44      0.44      0.44      3023

    accuracy                           0.92     44290
   macro avg       0.70      0.70      0.70     44290
weighted avg       0.92      0.92      0.92     44290

| TP:  39535 | FP:  1680 | TN:  1343 | FN:  1732
TPR:  0.9580294181791746
FPR:  0.555739331789613
TPR - FPR :  0.40229008638956165
```

Here, the difference between TPR and FPR was the largest among all (Model + features) set. After exporting all CSV files, in another experiment, I saw over 0.52 score for (TPR – FPR) score when one-hot encoded category features were added into Feature 3 as done in Feature 2.

>> CLUSTERING:

```
=======>>>OCSVM CLUSTER RESULT

Exporting LABEL...
Number of clusters: 2
\Label -1: 855255 data points
\Label 1: 15176 data points

DONE.
--------------------------------------
Wall time: 1min 36s
```



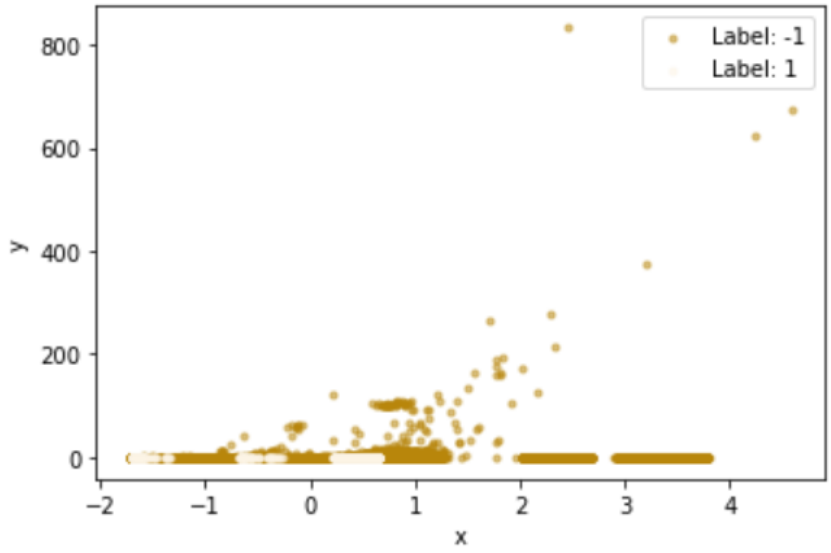
4) Iforest + Feature 1

>> SCORES:

```
========== IFOREST FEATURE 1 ==========
              precision    recall  f1-score   support

          -1       0.95      0.77      0.85     47581
           1       0.09      0.35      0.15      3182

    accuracy                           0.74     50763
   macro avg       0.52      0.56      0.50     50763
weighted avg       0.89      0.74      0.80     50763

| TP:  36566 | FP:  2065 | TN:  1117 | FN:  11015
TPR:  0.7685000315251886
FPR:  0.6489629164047769
TPR - FPR :  0.11953711512041176
```

>> Clustering:

```
=======>>>IFOREST CLUSTER RESULT

Exporting LABEL...
Number of clusters: 2
\Label -1: 1033881 data points
\Label 1: 19964 data points

DONE.
---------------------------------------
Wall time: 40.2 s
```



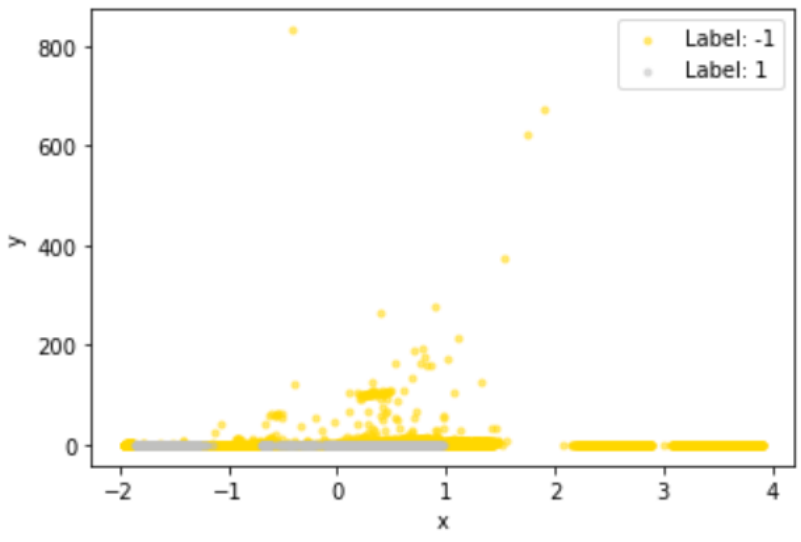
5) Iforest + Feature 2

>> SCORES:

```
=========== IFOREST FEATURE 2 ===========
              precision    recall  f1-score   support

          -1       0.94      0.94      0.94     47581
           1       0.12      0.12      0.12      3182

    accuracy                           0.89     50763
   macro avg       0.53      0.53      0.53     50763
weighted avg       0.89      0.89      0.89     50763

| TP:  44626 | FP:  2787 | TN:  395 | FN:  2955
TPR:  0.9378953784073475
FPR:  0.8758642363293526
TPR - FPR :  0.06203114207799487
```

>> CLUSTERING:

```
=========== IFOREST FEATURE 2 ===========
=======>>>IFOREST CLUSTER RESULT

Exporting LABEL...
Number of clusters: 2
\Label -1: 777845 data points
\Label 1: 276000 data points

DONE.
---------------------------------------
Wall time: 44.1 s
```



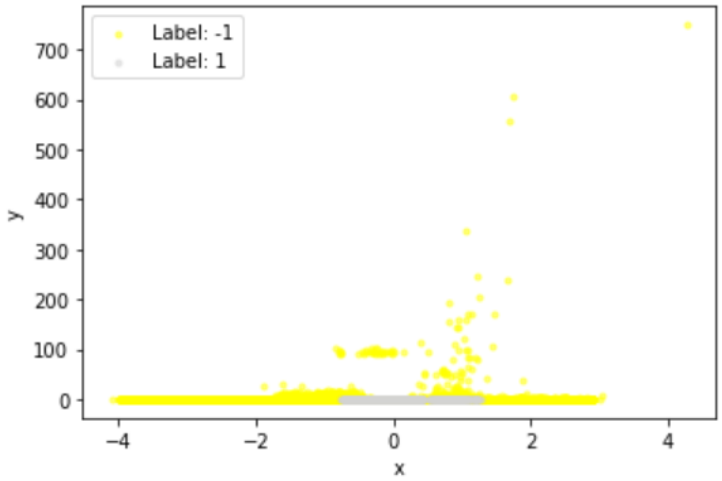
6) Iforest + Feature 3

>> SCORES:

```
=========== IFOREST FEATURE 3 ===========
              precision    recall  f1-score   support

          -1       0.93      0.99      0.96     41267
           1       0.13      0.02      0.03      3023

    accuracy                           0.92     44290
   macro avg       0.53      0.50      0.50     44290
weighted avg       0.88      0.92      0.90     44290

| TP:  40904 | FP:  2967 | TN:  56 | FN:  363
TPR:  0.9912036251726561
FPR:  0.9814753556070129
TPR - FPR :  0.00972826956564321
```

>> CLUSTERING:

```
=======>>>IFOREST CLUSTER RESULT

Exporting LABEL...
Number of clusters: 2
\Label -1: 858453 data points
\Label 1: 11978 data points

DONE.
--------------------------------------
Wall time: 30.9 s
```



Generally, accuracy is more or less the same and was not prioritized in the process of hyperparameter tuning. OCSVM was more robust in FPR than Iforest. Therefore, as seen above figures, OCSVM with feature 3 (Scaled cumulative numeric features grouped by stream_id + time-based feature + PCA) worked the best.

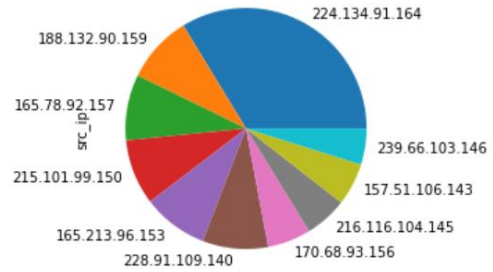
4.2 Interpretation of the result

As seen in the pie charts, one source IP address seems suspicious while the destination IP address is relatively evenly distributed.

1) OCSVM + feature1

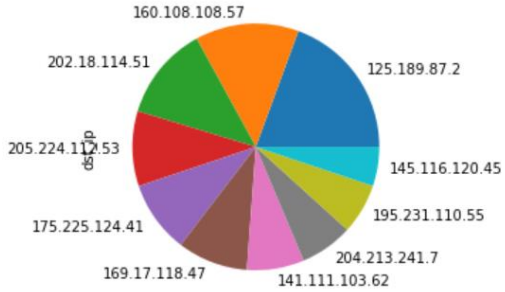
```
df_ocsvm_f1.src_ip.value_counts()[:10].plot.pie()
```

```
<AxesSubplot:ylabel='src_ip'>
```




```
df_ocsvm_f1.dst_ip.value_counts()[:10].plot.pie()
```

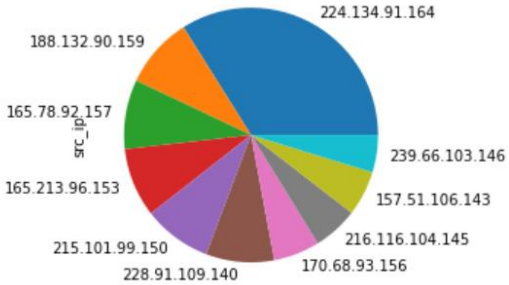
```
<AxesSubplot:ylabel='dst_ip'>
```



2) OCSVM + feature2

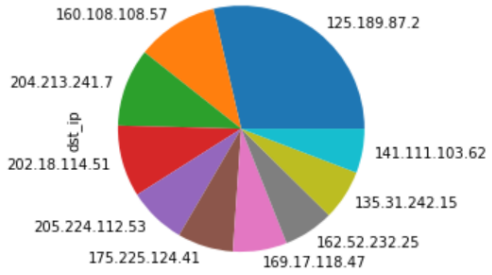
```
df_ocsvm_f2.src_ip.value_counts()[:10].plot.pie()
```

```
<AxesSubplot:ylabel='src_ip'>
```




```
df_ocsvm_f2.dst_ip.value_counts()[:10].plot.pie()
```

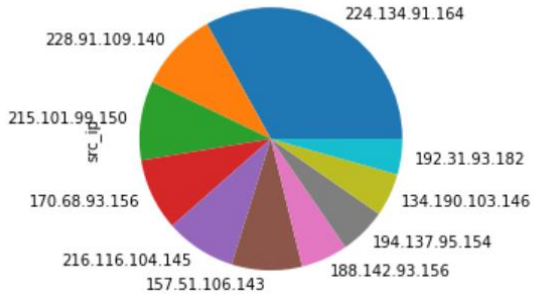
```
<AxesSubplot:ylabel='dst_ip'>
```



3) OCSVM + feature3

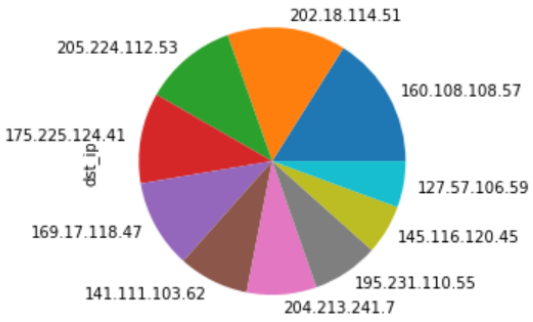
```
df_ocsvm_f3.src_ip.value_counts()[:10].plot.pie()
```

```
<AxesSubplot:ylabel='src_ip'>
```




```
df_ocsvm_f3.dst_ip.value_counts()[:10].plot.pie()
```

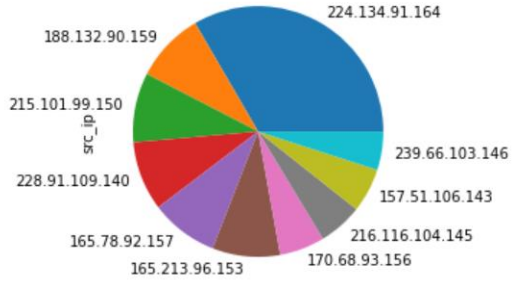
```
<AxesSubplot:ylabel='dst_ip'>
```



4) Iforest + feature 1

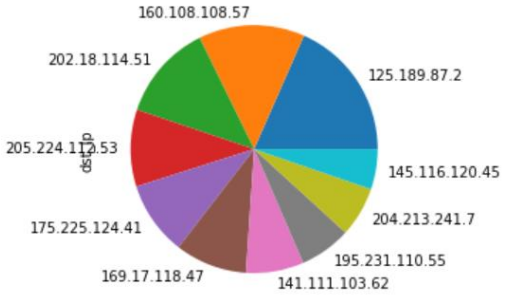
```
df_iforest_f1.src_ip.value_counts()[:10].plot.pie()
```

```
<AxesSubplot:ylabel='src_ip'>
```




```
df_iforest_f1.dst_ip.value_counts()[:10].plot.pie()
```

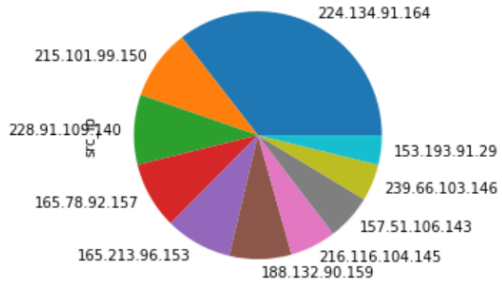
```
<AxesSubplot:ylabel='dst_ip'>
```



5) Iforest + feature 2

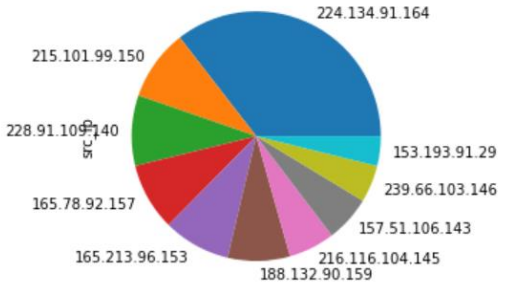
```
df_iforest_f2.src_ip.value_counts()[:10].plot.pie()
```

```
<AxesSubplot:ylabel='src_ip'>
```




```
df_iforest_f2.src_ip.value_counts()[:10].plot.pie()
```

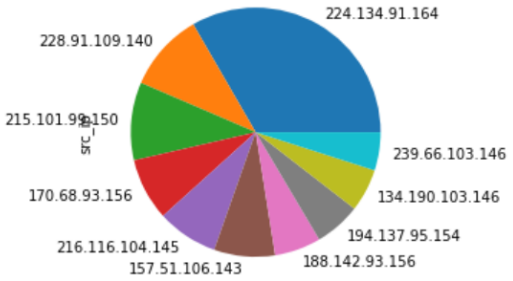
```
<AxesSubplot:ylabel='src_ip'>
```



6) Iforest + feature 3

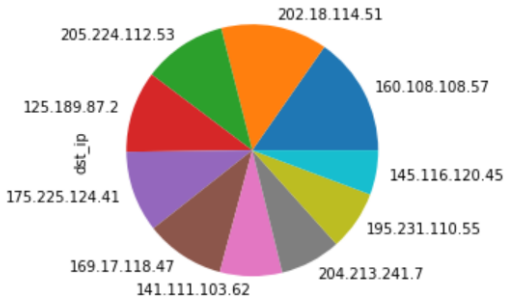
```
df_iforest_f3.src_ip.value_counts()[:10].plot.pie()
```

```
<AxesSubplot:ylabel='src_ip'>
```




```
df_iforest_f3.dst_ip.value_counts()[:10].plot.pie()
```

```
<AxesSubplot:ylabel='dst_ip'>
```



Among many, it seems very clear one IP address is outstanding compared to others: 224.134.91.164

The traffic coming from this address occurs much more than others.

While having this IP in mind, if querying by the unique conversation, we can find more interesting IPs.

```
In [333]: df_ocsvm_f2.flow_cnt.value_counts()[:10]

Out[333]: 224.134.91.164 ==> 125.189.87.2      68863
          224.134.91.164 ==> 204.213.241.7     33290
          215.101.99.150 ==> 162.52.232.25     21407
          228.91.109.140 ==> 135.31.242.15     21407
          153.193.91.29 ==> 125.189.87.2       15887
          228.91.109.140 ==> 182.141.104.3      7525
          215.101.99.150 ==> 217.145.94.97      7524
          165.78.92.157 ==> 232.19.91.2         7420
          165.213.96.153 ==> 184.141.93.4       7420
          188.132.90.159 ==> 148.205.90.101     7388
          Name: flow_cnt, dtype: int64
```

As seen in the above figure, 215.101.99.150 and 228.91.109.140 look suspicious occurring many times in other conversations too. Therefore, it is reasonable to assume these IP addresses are the attacker.

As seen in the first part of this report (1.2 Summary of attack), if we narrow down these IP address, the start and end of the attack by protocol are as follows. Here, src_ip is the attacker and dst_ip is the victim.

```
subission_ocsvm_f2.groupby(['src_ip', 'dst_ip', 'protocol']).agg(start=('starttime', 'min'), end=('endtime', 'max'))
```

			start	end
src_ip	**dst_ip**	**protocol**		
215.101.99.150	**162.52.232.25**	**tcp**	2012-12-05 01:24:32.928430000	2012-12-06 01:24:32.221493000
224.134.91.164	**125.189.87.2**	**icmp**	2012-12-02 15:03:04.407642000	2012-12-08 04:56:28.446218000
		udp	2012-12-01 15:44:12.046019000	2012-12-10 15:09:29.310224000
	204.213.241.7	**tcp**	2012-12-01 15:56:34.815956000	2012-12-08 11:22:03.115622000
228.91.109.140	**135.31.242.15**	**tcp**	2012-12-09 23:24:33.181435000	2012-12-10 23:24:32.097561000

6. Generating Adversarial samples

The task in this section is to generate adversarial samples by perturbing features of the training set so as to bypass detection. The dataset has the same features as previous sections but this time they have labels so we can apply supervised models.

6.1 Supervised model

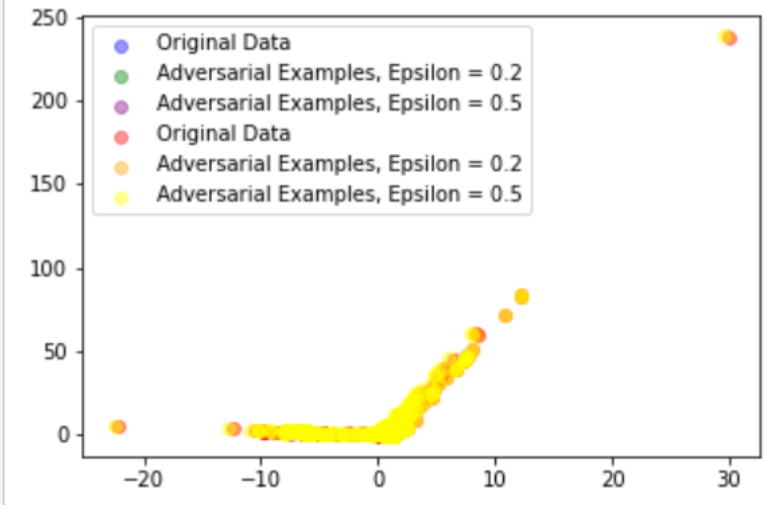
For the supervised model, Logistic regression was used. I used the reference code [4] and the code was customized not to use hyperparameters so in this case, I could not tune hyperparameters as done in earlier steps.

6.2 Fast Gradient Sign Method

FGSM (Fast Gradient Sign Method) was used for this task. FGSM is to add the noise towards the direction of the gradient of the cost function. The formula for this is X_ad = X + epsilon*signed Gradient. Epsilon is the paratmter which scales the noise [5]. Since the gradient works towards two directions, it is called Gradient Sign Method. I used the Feature3 in the previous section as this was the most robust among the features. Note that in the previous section, I used (feature 3 + OCSVM) with hyperparameter tuning with error rate less than 0.1 but this time Logistic regression without hyperparameter tuning was used so the error rate is quite high even without applying adversarial example

1) FGSM with epoch1

```
Epoch:  1
Error Rate without adversarial examples: 0.34964014507537117

Error Rate with adversarial examples, epsilon = 0.2: 0.9990932789300692
Error Rate with adversarial examples, epsilon = 0.5: 0.9990932789300692
```

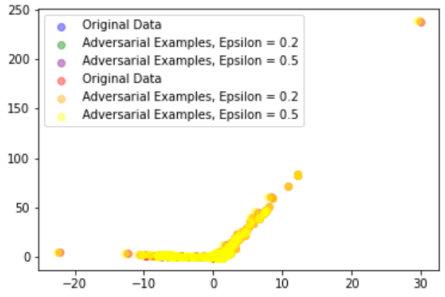


We can see that the adversarial example was very powerful. The error rate is almost 100%.

2) FGSM with epoch5

```
Epoch:  10
Error Rate without adversarial examples: 0.36776039895727075

Error Rate with adversarial examples, epsilon = 0.2: 0.5089963731157203
Error Rate with adversarial examples, epsilon = 0.5: 0.5089963731157203
```

```
C:\Users\kebin\Anaconda3\lib\site-packages\ipykernel_launcher.py:41: RuntimeWarning: overflow encountered in exp
```



However, as the epochs increases, the error rate decreased from 0.99 to 0.5.

7. Perturbing features and updating network

7.1 Manipulation of features via Gradient descent-based method

The feature was modified by FGSM discussed above. However, it was not easy to see how it was modified. The below picture is newly generated adversarial sample. It already looks pretty much the same as the original training data.

```
adv_df = pd.DataFrame(data=Xadv, columns=selected_features)
```

```
adv_df
```

bytes_bothdir	bytes_onedir	total_packets	duration	pps	bps_onedir	bpp_onedir	bps_bothdir	bpp_bothdir	cnt_timed_src	cnt_timed_dst	cnt_timed_sr
-0.239160	-0.226629	-0.241634	-0.411271	0.857007	-0.624921	0.581812	-0.836037	1.263197	-1.080234	-1.204642	-1.47
-0.239160	-0.226629	-0.241634	-0.411271	-0.075575	-0.624921	-0.097139	-0.836037	-0.101503	-1.080234	-1.204642	-1.47
-0.239160	-0.226629	-0.241634	-0.411271	-0.064796	-0.624921	-0.089291	-0.836037	-0.085729	-1.080234	-1.204642	-1.47
-0.239160	-0.226629	-0.241634	-0.411271	-0.069122	-0.624921	-0.092441	-0.836037	-0.092060	-1.080234	-1.204642	-1.47
-0.239160	-0.226629	-0.241634	-0.411271	-0.047980	-0.624921	-0.077048	-0.836037	-0.061121	-1.080234	-1.204642	-1.47
...	...	...	...	...	...	...	...	...	...	...	
-0.182166	-0.065463	-0.148792	-0.410621	-0.093707	5.178864	-0.105856	1.639140	-0.123185	-1.137013	-1.204642	-1.47
-0.182166	-0.065463	-0.148792	-0.410637	-0.093696	5.178864	-0.105733	1.639140	-0.123045	-1.137013	-1.204642	-1.47
-0.235009	-0.224132	-0.228371	-0.375802	-0.094151	-0.397497	-0.110662	0.109711	-0.128681	-1.137013	-1.203644	-1.47
-0.234831	-0.219785	-0.221739	-0.409902	-0.094103	0.121771	-0.110563	-0.229110	-0.128482	-1.137013	-1.203644	-1.47
-0.226609	-0.206003	-0.193887	-0.410843	-0.093796	0.483656	-0.109718	0.016602	-0.126828	-1.137013	-1.204642	-1.47

8. Conclusions

We discussed different methods of feature generation and 2 unsupervised models for anomaly detection. Performance of each feature applied to each model was shown and how the hyperparameters were tuned in each model was explained. Contrary to project 1, where explicit feature were the most prominent features, more new features in anomaly detection were created using existing data and applied to anomaly detection models. Finally, Adversarial example was generated successfully using FGSM.

[Reference]

1. F. T. Liu, K. M. Ting, and Z.-H. Zhou, “Isolation-based anomaly detection,” ACM

Trans. Knowl. Discov. Data, vol. 6, no. 1, pp. 3:1–3:39, Mar. 2012. [Online]. Available:

http://doi.acm.org.ezp.lib.unimelb.edu.au/10.1145/2133360.2133363

2. sklearn One class SVM (https://scikit-learn.org/stable/modules/generated/sklearn.svm.OneClassSVM.html?highlight=oneclass%20svm#sklearn.svm.OneClassSVM)

3. sklearn IsolationForest (https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html)

4. Generating adversarial samples (https://github.com/kenhktsui/adversarial_examples/blob/master/adversarial.py)

5. Fast Gradient Sign Method (https://towardsdatascience.com/perhaps-the-simplest-introduction-of-adversarial-examples-ever-c0839a759b8d)