Progess of the Project

Tsung-Min Pai

2023/8/16

Outline

- **GNN -** Graph Attention Network
  - Dataset
  - Observation
  - Modification
    - Repeat time
    - Model
    - Data Format
- **Future Work**

GNN

Data Format

- Old Version:


```
{"label": 10, "num_nodes": 3, "node_feat": [205565, 733769, 250773], "edge_attr": [23, 23], "edge_index": [[0, 0], [1, 2]]}
{"label": 11, "num_nodes": 3, "node_feat": [470650, 663446, 627322], "edge_attr": [23, 23], "edge_index": [[0, 0], [1, 2]]}
{"label": 15, "num_nodes": 2, "node_feat": [9863, 103498], "edge_attr": [23], "edge_index": [[0], [1]]}
{"label": 16, "num_nodes": 2, "node_feat": [157277, 753159], "edge_attr": [23], "edge_index": [[0], [1]]}
{"label": 22, "num_nodes": 36, "node_feat": [83068, 614681, 444724, 266227, 121794, 623948, 116790, 769462, 255741, 169794,
{"label": 0, "num_nodes": 7, "node_feat": [150942, 396371, 507529, 763634, 318841, 126686, 88192], "edge_attr": [11, 19, 15, 25,
{"label": 0, "num_nodes": 3, "node_feat": [264800, 229960, 554967], "edge_attr": [19, 15], "edge_index": [[0, 0], [1, 2]]}
{"label": 0, "num_nodes": 4, "node_feat": [416286, 466370, 94352, 15536], "edge_attr": [9, 9, 9, 9], "edge_index": [[0, 2, 0, 2],
```


  - Train, Validation, Test all in this format → graph

- New Version:


```
{"label": 103, "num_nodes": 2, "node_feat": [249632, 545864], "edge_attr": [1], "edge_index": [[0], [1]]}
{"label": 76, "num_nodes": 2, "node_feat": [562981, 268631], "edge_attr": [21], "edge_index": [[0], [1]]}
{"label": 76, "num_nodes": 2, "node_feat": [562981, 163190], "edge_attr": [21], "edge_index": [[0], [1]]}
```


  - Validation, Test in this format(triplet → subgraph)
  - Triplets woud not repeat

Data Format

- Original data format:


```
{"label": 10, "num_nodes": 3, "node_feat": [205565, 733769, 250773], "edge_attr": [23, 23], "edge_index": [[0, 0], [1, 2]]}
{"label": 11, "num_nodes": 3, "node_feat": [470650, 663446, 627322], "edge_attr": [23, 23], "edge_index": [[0, 0], [1, 2]]}
{"label": 15, "num_nodes": 2, "node_feat": [9863, 103498], "edge_attr": [23], "edge_index": [[0], [1]]}
{"label": 16, "num_nodes": 2, "node_feat": [157277, 753159], "edge_attr": [23], "edge_index": [[0], [1]]}
{"label": 22, "num_nodes": 36, "node_feat": [83068, 614681, 444724, 266227, 121794, 623948, 116790, 769462, 255741, 169794,
```

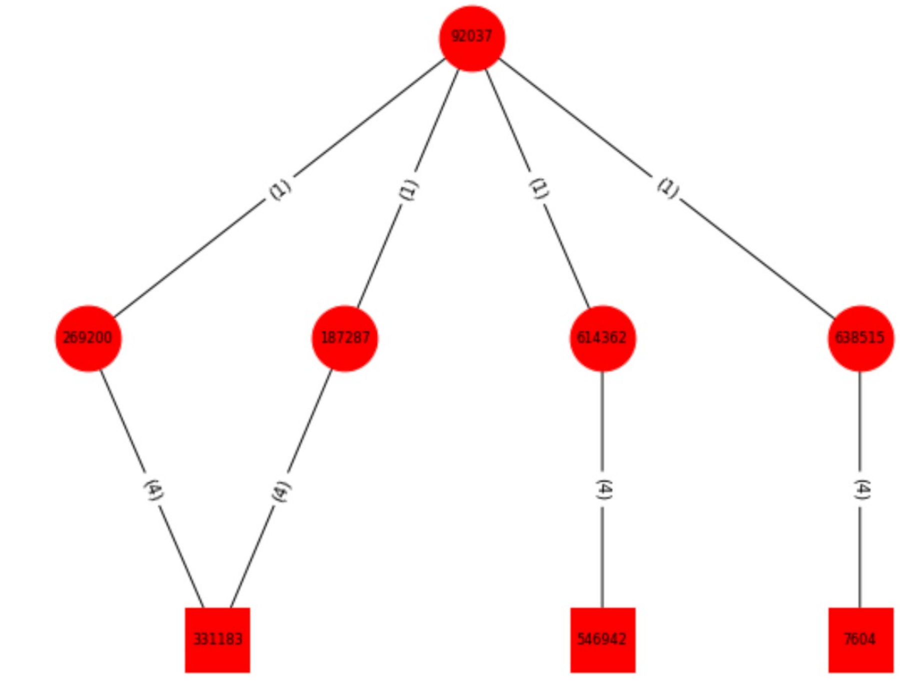

- Map the **node_feat** to its **embedding** based on the **transR_50.vec.json**
  - node id correspond to a vector of **50-dim**

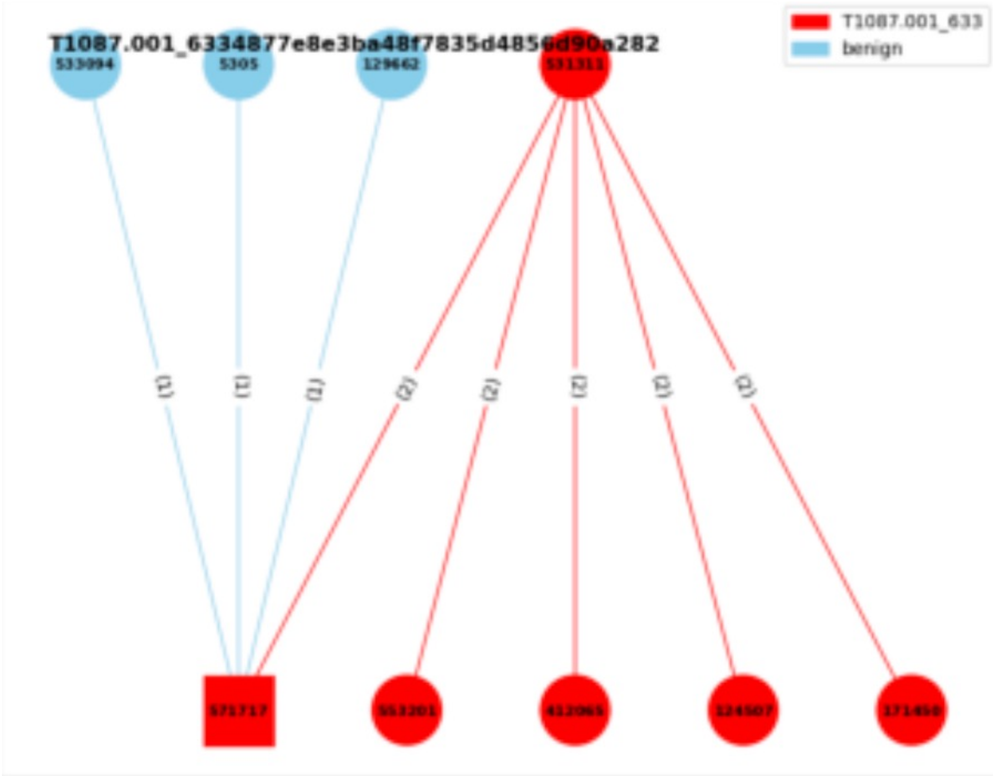

```
"node_feat": [[-0.004259330220520496, 0.02023318223655224, 0
37990725412965, -0.01702643744647503, 0.0013664175057783723,
65514039993, -0.00885914359241724, -0.0010546729899942875, -
9137960374355, -0.009182648733258247, -0.008827704936265945,
9938482046127, -0.0014976661186665297, 0.008676833473145962,
```


- We have the maps recording the:
  1. **label** to original **name** of AP
  2. **Real node id** to the **DGL** (Deep Graph Library) node id

Data Format

Current version:





Graph Attention Network - GAT

Graph Attention Network - GAT

Model:

```
class GAT(nn.Module):
    def __init__(self, in_dim, hidden_dim, out_dim, num_heads, dropout_prob=0.2):
        super(GAT, self).__init__()
        # do not check the zero in_degree since we have all the complete graph
        self.layer1 = GATConv(in_dim, hidden_dim, num_heads=num_heads, activation=F.relu, allow_zero_in_degree=True)
        self.layer2 = GATConv(hidden_dim * num_heads, out_dim, num_heads=num_heads, allow_zero_in_degree=True)

        # Adding Dropout for regularization
        self.dropout = nn.Dropout(dropout_prob)

    def forward(self, g, h):
        # Apply GAT layers
        h = self.layer1(g, h)
        h = h.view(h.shape[0], -1)
        h = F.relu(h)
        h = self.dropout(h)
        h = self.layer2(g, h).squeeze(1)

        # Store the output as a new node feature
        g.ndata['h_out'] = h

        # Use mean pooling to aggregate this new node feature
        h_agg = dgl.mean_nodes(g, feat='h_out')
        return h_agg
```

- Use the **new** verison of the dataset

Observation

- While batch size = 1:
  - # of repeat time small → bad performance

- 10 APs x 50 times:

```
labels: tensor([57], device='cuda:1') torch.Size([1])
predicted: tensor([74], device='cuda:1') torch.Size([1])
labels: tensor([0], device='cuda:1') torch.Size([1])
predicted: tensor([74], device='cuda:1') torch.Size([1])
Test Accuracy: 10 %
```

→ All guess 74

- 165 APs x 5 times:

```
labels: tensor([73], device='cuda:1') torch.Size([1])
predicted: tensor([22], device='cuda:1') torch.Size([1])
labels: tensor([139], device='cuda:1') torch.Size([1])
predicted: tensor([146], device='cuda:1') torch.Size([1])
labels: tensor([83], device='cuda:1') torch.Size([1])
predicted: tensor([146], device='cuda:1') torch.Size([1])
Test Accuracy: 3 %
```

→ 92.7% guess 74
(153/165)

- AP + benign x 50 times:

```
labels: tensor([38], device='cuda:0') torch.Size([1])
predicted: tensor([0], device='cuda:0') torch.Size([1])
labels: tensor([145], device='cuda:0') torch.Size([1])
predicted: tensor([145], device='cuda:0') torch.Size([1])
labels: tensor([96], device='cuda:0') torch.Size([1])
predicted: tensor([0], device='cuda:0') torch.Size([1])
Test Accuracy: 26 %
```

→ 74% guess benign
(1480/2000)

Prerequesite

- Try to find where the problem comes from
  - Dataset, Data Format, Model, Training Code…

- Fix the random seed , initial weight, seqeunce and learning rate

```
seed = 8787
same_seeds(seed)

model = GAT(in_dim=50, hidden_dim=16, out_dim=168, num_heads=8)
torch.save(model.state_dict(), 'model1_initial/initial_weight.pth')
```

```
model.layer1.fc.weight
```

```
Parameter containing:
tensor([[-0.1806, -0.0598,  0.0091,  ...,  0.0719,  0.2496,  0.0873],
        [ 0.1694, -0.0015, -0.0139,  ...,  0.0147,  0.0892,  0.0146],
        [ 0.0969, -0.0595, -0.0115,  ..., -0.0474,  0.0529, -0.0565],
        ...,
        [-0.0433, -0.2248,  0.3002,  ...,  0.0850,  0.1621,  0.0422],
        [ 0.2097, -0.2492,  0.0612,  ..., -0.0041,  0.0365, -0.1483],
        [ 0.0971, -0.2221,  0.1652,  ..., -0.1312, -0.2610,  0.0077]],
       requires_grad=True)
```

```
def create_dataloaders(batch_size, shuffle=False):
    dataloaders = {}
    for dataset_name, dataset in dataset_data.items():
        dataloaders[dataset_name] = DataLoader(dataset, batch_size=batch_size, shuffle=shuffle, collate_fn=collate)
    return dataloaders
```

Modification – Repeat Time

- While batch size is small or the training data repeat many times:

```
labels: tensor([128], device='cuda:1') torch.Size([1])
predicted: tensor([128], device='cuda:1') torch.Size([1])
labels: tensor([122], device='cuda:1') torch.Size([1])
predicted: tensor([122], device='cuda:1') torch.Size([1])
labels: tensor([120], device='cuda:1') torch.Size([1])
predicted: tensor([120], device='cuda:1') torch.Size([1])
Test Accuracy: 100 %
```

→ Test Accuracy = 1

(5 APs x 500 times, batch size = 1)

```
100%|██████████| 255000/255000 [1:26:43<00:00, 49.00it/s]
 80%|████████  | 4/5 [5:39:39<1:25:33, 5133.80s/it]

total count: 255000
Epoch 3 | Train Loss: 0.8456 | Train Accuracy: 0.7129
Validation Loss: 0.8838 | Validation Accuracy: 0.6923
```

→ Validation Accuracy = 0.6923

(51 APs x 20000 times, batch size = 4)

- Epoch is small
- Training data appear in validation and testing data → check the capability of derectly remembering the graph
- Data probably is not the problem causing the bad performance
    - suspected that maybe some APs’ embedding is far away from others

Modification – Model

```
class GAT(nn.Module):
    def __init__(self, in_dim, hidden_dim, out_dim, num_heads, dropout_prob=0.25):
        super(GAT, self).__init__()
        self.layer1 = GATConv(in_dim, hidden_dim, num_heads=num_heads, activation=F.relu, allow_zero_in_degree=True)
        self.layer2 = GATConv(hidden_dim * num_heads, hidden_dim, num_heads=1, allow_zero_in_degree=True)
        self.layer3 = GATConv(hidden_dim, out_dim, num_heads=1, allow_zero_in_degree=True)
        self.dropout = nn.Dropout(dropout_prob)

    def forward(self, g, h):
        h1 = self.layer1(g, h)
        h1 = h1.view(h1.shape[0], -1)
        h1 = F.relu(h1)
        h1 = self.dropout(h1)

        h2 = self.layer2(g, h1)
        h2 = h2.view(h2.shape[0], -1)
        h2 = F.relu(h2)
        h2 = self.dropout(h2)

        h3 = self.layer3(g, h2)
        h3 = self.dropout(h3)

        # Store the output as a new node feature
        g.ndata['h_out'] = h3

        # Use mean pooling to aggregate this new node feature
        h_agg = dgl.mean_nodes(g, feat='h_out')
        return h_agg
```

- Performance is worse then model 1 with the same dataset, repeat time and batch size
  - Maybe need more training steps

Modification – Data Format

- Use the new format:
  - training data do not appear in the validation and testing data
  - Validation and testing data are triplets not in the training graph → extract from the original graph

```
08/15/2023, 23:10:22# labels of 5000: tensor([84, 82, 86, 87], device='cuda:0') torch.Size([4])
08/15/2023, 23:10:22# predicted of 5000: tensor([78, 76, 84, 78], device='cuda:0') torch.Size([4])
08/15/2023, 23:11:46# labels of 10000: tensor([ 76,  70,  71, 119], device='cuda:0') torch.Size([4])
08/15/2023, 23:11:46# predicted of 10000: tensor([ 76,  79,  76, 119], device='cuda:0') torch.Size([4])
08/15/2023, 23:13:10# labels of 15000: tensor([87, 88, 90, 89], device='cuda:0') torch.Size([4])
```

```
08/15/2023, 23:25:13# Epoch 31 | Train Loss: 2.5340 | Train Accuracy: 0.1510
08/15/2023, 23:25:13# labels of False: tensor([ 71,  71, 119,  77], device='cuda:0') torch.Size([4])
08/15/2023, 23:25:13# predicted of False: tensor([100, 100, 100, 100], device='cuda:0') torch.Size([4])
08/15/2023, 23:25:13# labels of False: tensor([ 82, 103, 103,  76], device='cuda:0') torch.Size([4])
08/15/2023, 23:25:13# predicted of False: tensor([100, 100, 100, 100], device='cuda:0') torch.Size([4])
08/15/2023, 23:25:13# labels of False: tensor([76], device='cuda:0') torch.Size([1])
08/15/2023, 23:25:13# predicted of False: tensor([100], d   e='cuda:0') torch.Size([1])
08/15/2023, 23:25:13# Validatio   ccuracy: 0.0000
```

Ps x 5000 times, batch size = 4)

```
08/15/2023, 23:17:06# Epoch 30 | Train Loss: 2.5343 | T   Accuracy: 0.1522
08/15/2023, 23:17:06# labels of False: tensor([ 71,  71, 119,  77], device='cuda:0') torch.Size([4])
08/15/2023, 23:17:06# predicted of False: tensor([85, 85, 85, 85], device='cuda:0') torch.Size([4])
08/15/2023, 23:17:06# labels of False: tensor([ 82, 103, 103,  76], device='cuda:0') torch.Size([4])
08/15/2023, 23:17:06# predicted of False: tensor([85, 85, 85, 85], device='cuda:0') torch.Size([4])
08/15/2023, 23:17:06# labels of False: tensor([76], device='cuda:0') torch.Size([1])
08/15/2023, 23:17:06# predicted of False: tensor([85], device='cuda:0') torch.Size([1])
08/15/2023, 23:17:06# Validation Loss: 5.0077 | Validation Accuracy: 0.0000
```

(23 APs x 5000 times, batch size = 4)

Modification – Data Format

- With new version(triplet) of dataset
  - But training data would be in the validation and testing data
  - Not shuffle but the predction of validation is different
  - Training accuracy slowly increase → 6h, from 0.05 to 0.13

```
labels of Validation: tensor([ 71,  71, 119,  77],
predicted of Validation: tensor([90, 90, 90, 90],
labels of Validation: tensor([ 82, 103, 103,  76],
predicted of Validation: tensor([90, 90, 90, 90],
```

→ Validation Accuracy = 0

Epoch 30

```
labels of Validation: tensor([ 71,  71, 119,  77],
predicted of Validation: tensor([87, 87, 87, 87],
labels of Validation: tensor([ 82, 103, 103,  76],
predicted of Validation: tensor([87, 87, 87, 87],
```

→ Validation Accuracy = 0

Epoch 32

Future Work

Future Work

- **GNN**
  - Figure out the reason causing the currently bad performance on both GCN, GAT
  - Read some paper about these GNN models in classification
  - Try the **GraphSAGE**
- **Graphormer** (if available)
  - Write the trainer (training part)

Thanks!!

Graph Convolutional Network - GCN

Graph Convolutional Network - GCN

Model:

```python
class GCN(nn.Module):
    def __init__(self, in_feats, hidden_size, num_classes):
        super(GCN, self).__init__()
        self.conv1 = GraphConv(in_feats, hidden_size)
        self.conv2 = GraphConv(hidden_size, num_classes)

    def forward(self, g, inputs):
        h = self.conv1(g, inputs)
        h = torch.relu(h)
        h = self.conv2(g, h)

        g.ndata['h'] = h
        hg = dgl.mean_nodes(g, 'h')
        return hg
```

- Use the **old** verison of the dataset
- Use **DGL** to be our library
- DGL data format:

```
batched_g is like:
Graph(num_nodes=96, num_edges=160, ndata_schemes={'feat': Scheme(shape=(1,), dtype=torch.int64)}, edata_schemes={})
num_nodes = 3*batch_size, num_edges = 5*batch_size

labels is like: tensor([ 76,   0,   0,   0,   0,   0,   0,   0,   0,  76,   0,  76,   0,   0,
                          0,   0,  76,   0,  30,  92,   0,   0,  76,   0,   0,   0,   0,   0,
                        116,   0,  76,  76])
```

Result:

```
  0%|                    | 0/120 [00:00<?, ?it/s]
Epoch 0 | Train Loss: 2625.5943 | Train Accuracy: 0.4763
  1%|                    | 1/120 [00:56<1:52:21, 56.65s/it]
Validation Loss: 494.0275 | Validation Accuracy: 0.6642
 99%|███████████████████| 119/120 [1:51:06<00:55, 55.13s/it]
Validation Loss: 0.9964 | Validation Accuracy: 0.6642
Epoch 119 | Train Loss: 0.9625 | Train Accuracy: 0.6644
100%|███████████████████| 120/120 [1:52:03<00:00, 56.03s/it]
Validation Loss: 0.9965 | Validation Accuracy: 0.6642

Test Accuracy: 66 %
```

- GAT applied on the old data has the similar result

TRAM

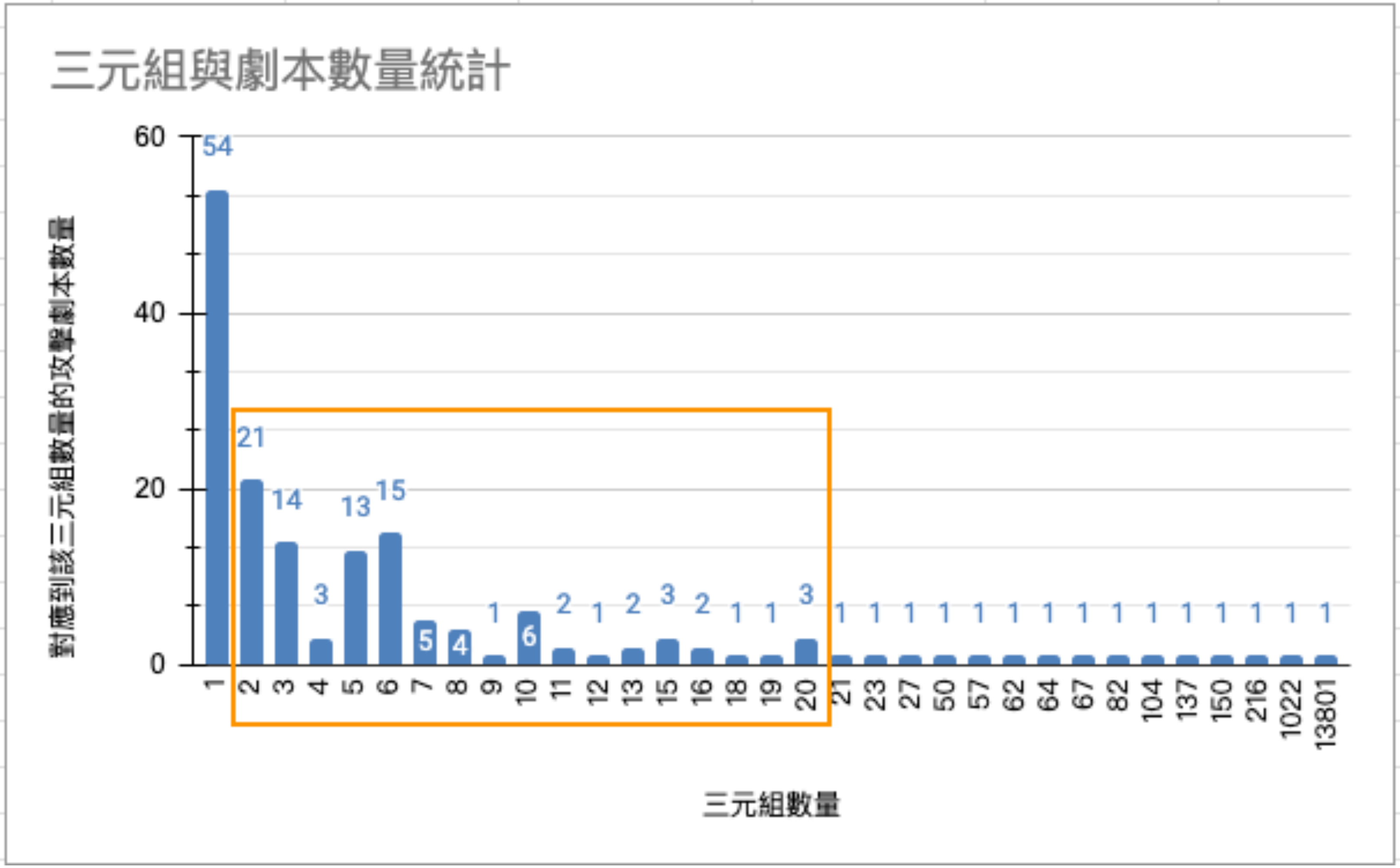
Automation

Job: Analyze Malware-Madness-EXCEPTION-edition.pdf By: djangoSuperuser on 2023-23-25 15:23:16 UTC		Error	
Job: Analyze Hive-Analysis-Study.pdf By: djangoSuperuser on 2023-23-25 15:23:16 UTC		Error	
Bootstrap Training Data By: pipeline (manual) on 2022-06-04 01:05:13 UTC	Analyze / Export	Accepted	Accepted: 12588 Reviewing: 0 Total: 12588
Report for MOLERATS-IN-THE-CLOUD-New-Malware-Arsenal-Abuses-Cloud-Platf.pdf By: djangoSuperuser on 2023-07-25 15:22:16 UTC	Analyze / Export / Download	Reviewing	Accepted: 0 Reviewing: 112 Total: 112
Report for Suspected-Iran-Nexus-TAG-56-Uses-UAE-Forum-Lure-for-Credenti.pdf By: djangoSuperuser on 2023-07-25 15:22:24 UTC	Analyze / Export / Download	Reviewing	Accepted: 0 Reviewing: 120 Total: 120

- Successfully **upload** the pdf files
- Successfully **export** the pdf files
  - Click 3 times and then scroll $\frac{1}{3}$ of the window size

```python
if count % 3 == 0:
    driver.execute_script(f"window.scrollBy(0, {window_height/3});")
    time.sleep(1)
```

Appendix


三元組與劇本數量統計
對應到該三元組數量的攻擊劇本數量
60
40
20
0
54
21
14
3
13
15
5
4
1
6
2
1
2
3
2
1
1
3
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 2 3 4 5 6 7 8 9 10 11 12 13 15 16 18 19 20 21 23 27 50 57 62 64 67 82 104 137 150 216 1022 13801
三元組數量