Progess of the Project

Tsung-Min Pai

2023/12/08

Outline

- **Recap**
  - **Input Format**
  - **Model Design**
  - **Difficulty**

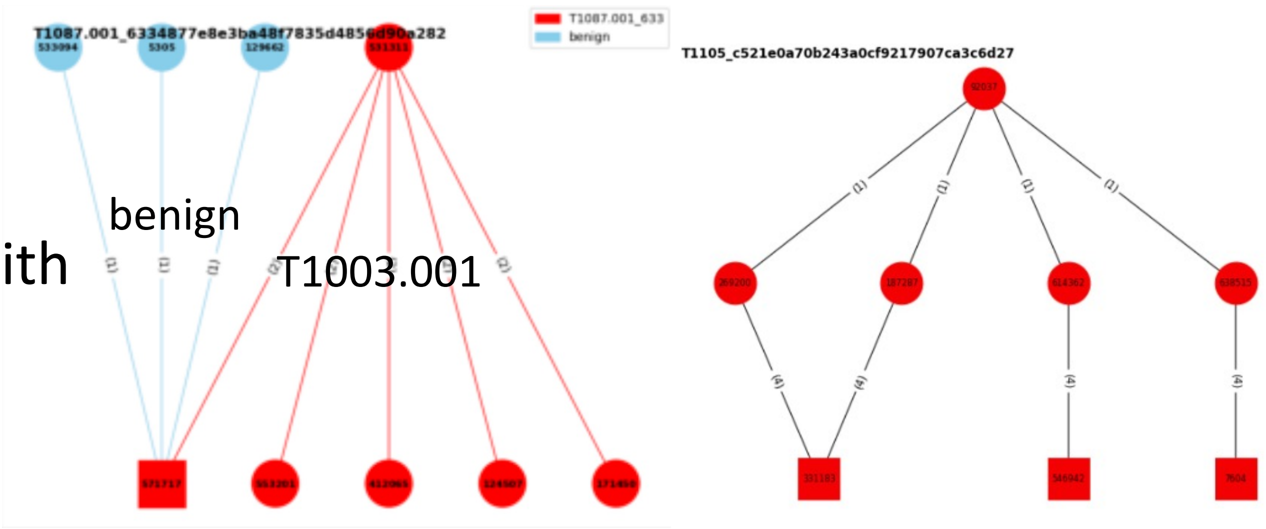
- **Experiments**
  - **Oversampling**
  - **Change the Dataset**
  - **Predict More Labels**
  - **DARPA Format**

- **Future Work**

Recap

Input Format

- Each data is a graph → label the triplets with the benign or the specific AP
    - Consider the nodes and the edges



Format in experiment 3:

```
{"labels": [45, 65, 45, 45], "num_nodes": 4, "node_feat": [578353, 695633, 234474, 883199], "edge_attr": [24, 2, 7, 2],
{"labels": [45, 65, 45, 45], "num_nodes": 4, "node_feat": [578353, 234474, 1085219, 1079260], "edge_attr": [24, 2, 7, 2]
{"labels": [45, 65, 45, 45], "num_nodes": 4, "node_feat": [578353, 946954, 234474, 391415], "edge_attr": [24, 2, 7, 2],
```

- **Edge classification**
- # of labels = # of edges

Source txt file:

```
853776 595218 13 a
593289 563219 17 b
388326 563219 17 b
```

- **a** means attack pattern
- **b** means benign

- This is the same format how I handle the **DARPA** dataset now

Model Design

```
g.ndata['feat'] = th.tensor(data["node_feat"])
g.edata['feat'] = th.tensor(data["edge_attr"])
g.edata['label'] = th.tensor(data["labels"])
```

- Concept from the DGL official website:
  1. Let the dgl graph's edge data have the attribute: **edata["label"]**
  2. Use **GraphSAGE** model to get the new **node embedding**
  3. Use **MLP** model to get the '**score**' of the **edge**
  4. **Concatenate** these two models
  5. Train the final model

```
class Model(nn.Module):
    def __init__(self, in_features, hidden_features, out_features, num_classes):
        super().__init__()
        self.sage = GraphSAGE(in_features, hidden_features, out_features)
        self.pred = MLPPredictor(out_features, num_classes)

    def forward(self, g, node_feat, return_logits=False):
        h = self.sage(g, node_feat)
        logits = self.pred(g, h)

        return logits
```

Model Design

1. Let the dgl graph's edge data have the attribute: **edata["label"]**
2. Use **GraphSAGE** model to get the new **node embedding**
3. Use **MLP** model to get the **'score'** of the **edge**
4. **Concatenate** these two models
5. Train the final model

Originally is the embedding based on the Trans Family or SecureBert
→ After step2. all the nodes' embedding would be updated

```
class GraphSAGE(nn.Module):
    def __init__(self, in_dim, hidden_dim, out_dim):
        super(GraphSAGE, self).__init__()
        self.layer1 = dglnn.SAGEConv(in_dim, hidden_dim, 'pool')
        self.layer2 = dglnn.SAGEConv(hidden_dim, out_dim, 'pool')

    def forward(self, g, inputs):
        h = self.layer1(g, inputs)
        h = torch.relu(h)
        h = self.layer2(g, h)
        return h
```

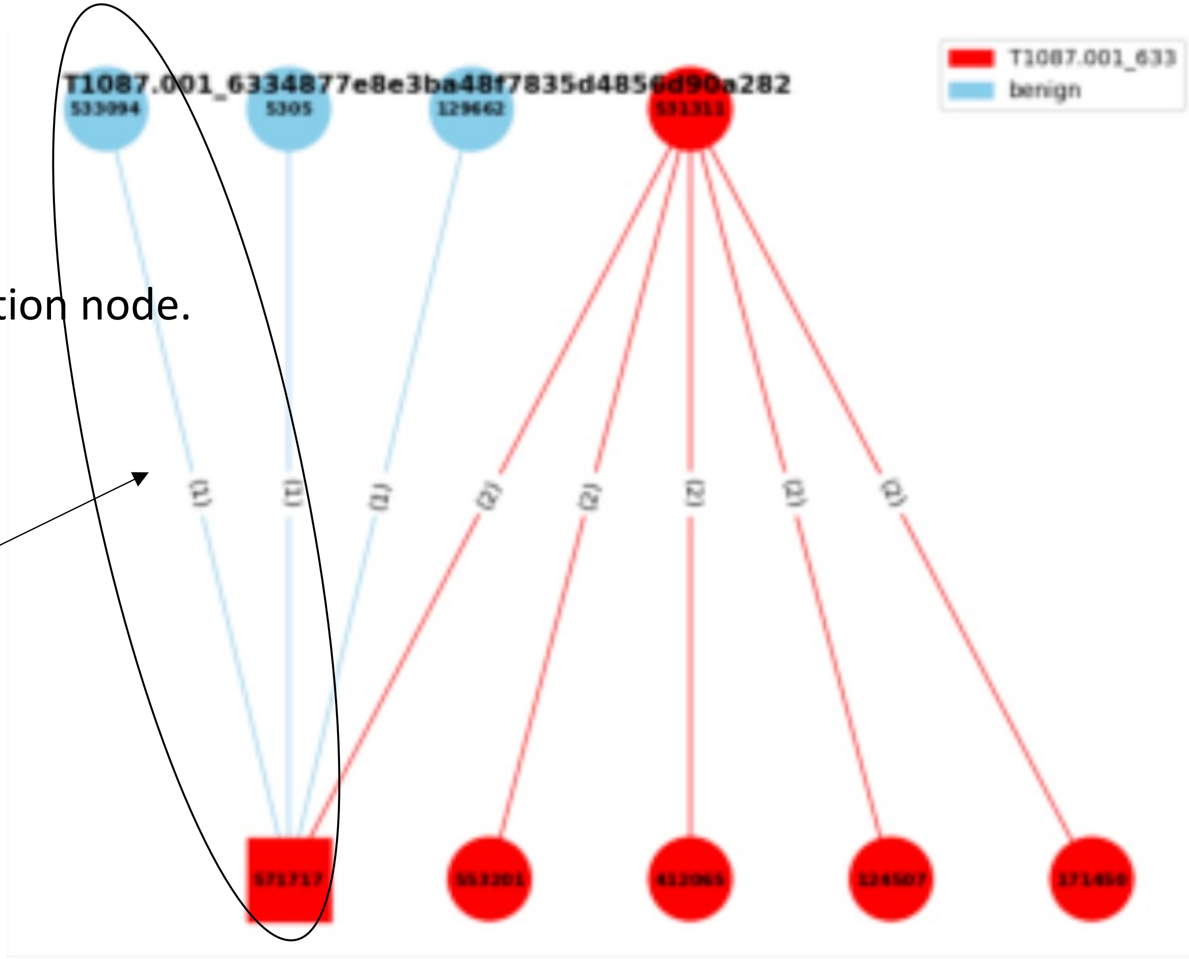
Model Design

1. Let the dgl graph's edge data have the attribute: **edata["label"]**
2. Use **GraphSAGE** model to get the new **node embedding**
3. Use **MLP** model to get the **'score'** of the **edge**
4. **Concatenate** these two models
5. Train the final model

Each time would choose an edge and get the source and destination node.
→ Use updated node embedding to 'score' the edge

```
class MLPPredictor(nn.Module):
    def __init__(self, out_feats, out_classes):
        super().__init__()
        self.W = nn.Linear(out_feats*2, out_classes)

    def apply_edges(self, edges):
        h_u = edges.src['h']
        h_v = edges.dst['h']
        score = self.W(torch.cat([h_u, h_v], 1))
        return {'score': score}

    def forward(self, graph, h):
        with graph.local_scope():
            graph.ndata['h'] = h
            graph.apply_edges(self.apply_edges)
            return graph.edata['score']
```

Update the node embedding



Model Design

1. Let the dgl graph's edge data have the attribute: **edata["label"]**
2. Use **GraphSAGE** model to get the new **node embedding**
3. Use **MLP** model to get the '**score**' of the edge
4. **Concatenate** these two models
5. Train the final model

```
class Model(nn.Module):
    def __init__(self, in_features, hidden_features, out_features, num_classes):
        super().__init__()
        self.sage = GraphSAGE(in_features, hidden_features, out_features)
        self.pred = MLPredictor(out_features, num_classes)

    def forward(self, g, node_feat, return_logits=False):
        h = self.sage(g, node_feat)
        logits = self.pred(g, h)

        return logits
```

Get the new node embedding

Use the new node embedding to predict the edge

Batched graph we feed into the model

```
def model_fn(batched_g, model, criterion, device, count=1, which_type='train'):
    """Forward a batch through the model."""
    batched_g = batched_g.to(device)
    labels = batched_g.edata['label'].to(device)

    logits = model(batched_g, batched_g.ndata['feat'].float())
    loss = criterion(logits, labels)

    output = torch.softmax(logits, dim=1)
    preds = output.argmax(1)

    accuracy = torch.mean((preds == labels).float())
```

I do not use the original edge embedding → may be a problem?

Difficulty

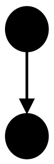
- Current Problem:
  - Can’t predict the edge in the small graphs consist of single triplet

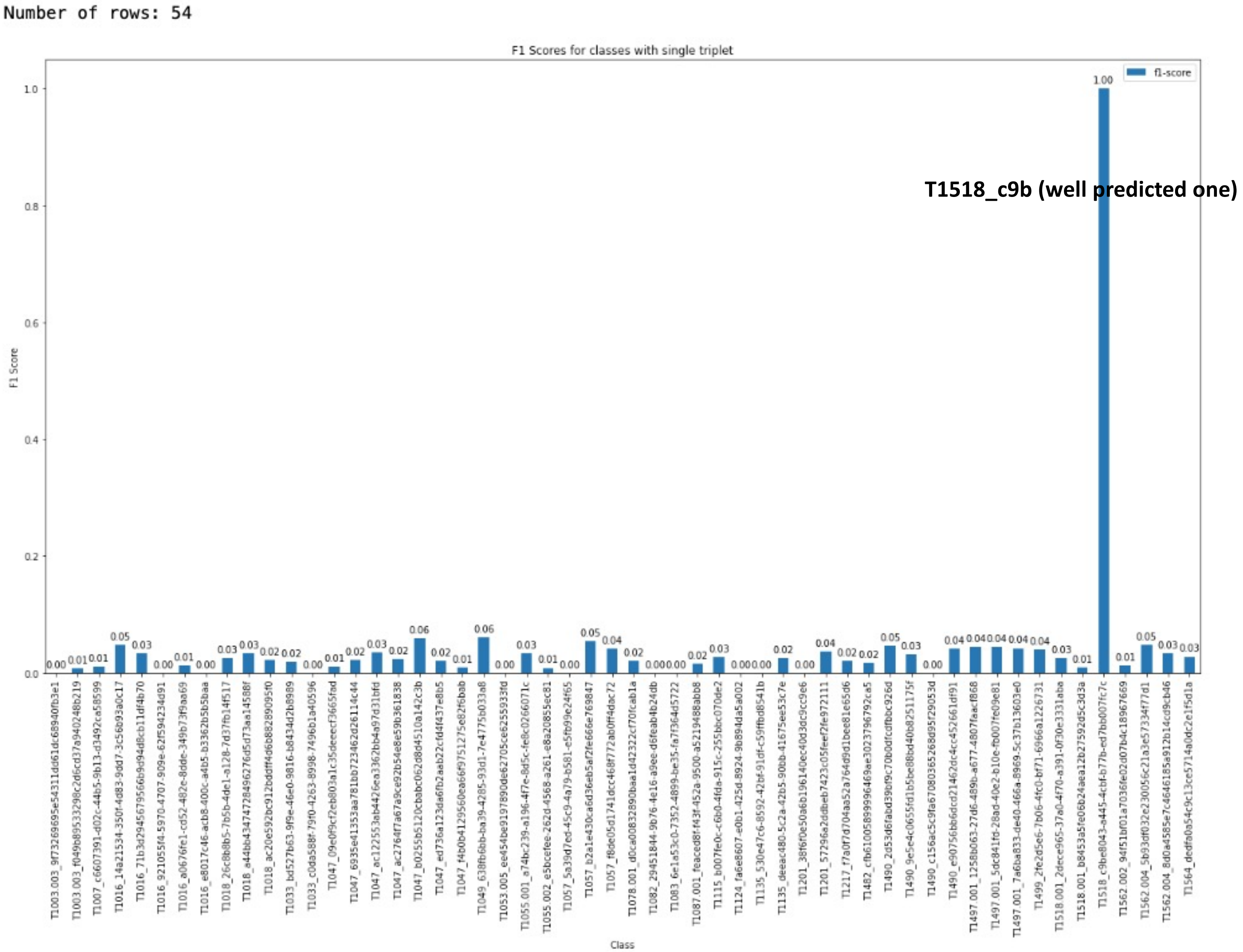
	precision	recall	f1-score	support
T1003.003_9f73269695e54311dd61dc68940fb3e1	0.0	0.0	0.0	100.0
T1003.003_f049b89533298c2d6cd37a940248b219	0.0	0.0	0.0	100.0
T1007_c6607391-d02c-44b5-9b13-d3492ca58599	0.0	0.0	0.0	100.0
T1016_14a21534-350f-4d83-9dd7-3c56b93a0c17	0.0	0.0	0.0	100.0
T1016_71b3d2945679566b9d94d8cb11df4b70	0.0	0.0	0.0	100.0
T1016_921055f4-5970-4707-909e-62f594234d91	0.0	0.0	0.0	100.0
T1016_a0676fe1-cd52-482e-8dde-349b73f9aa69	0.0	0.0	0.0	100.0

```
Number of support=100: 54
Number of support=100 and f1-score<=0.2: 53
Number of support=100 and f1-score=0: 48
Number of support=200 and f1-score=0: 6
Number of support>200 and f1-score=0: 0
```

Difficulty

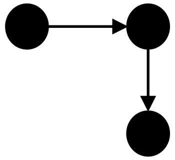
Prediction of single triplet case:



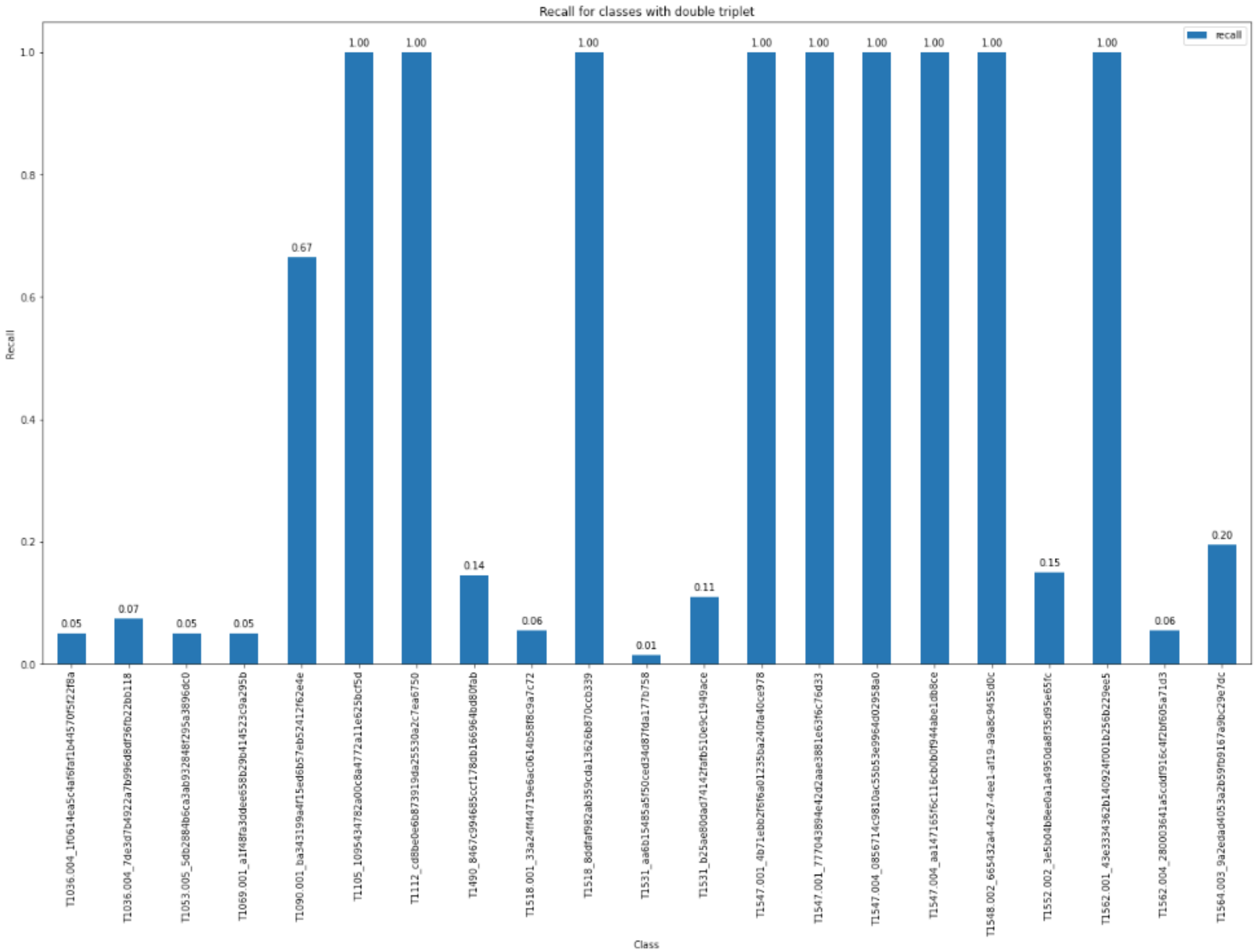


Difficulty

Prediction of double triplet case:

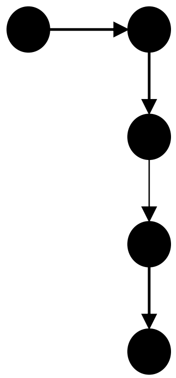


Number of rows: 21

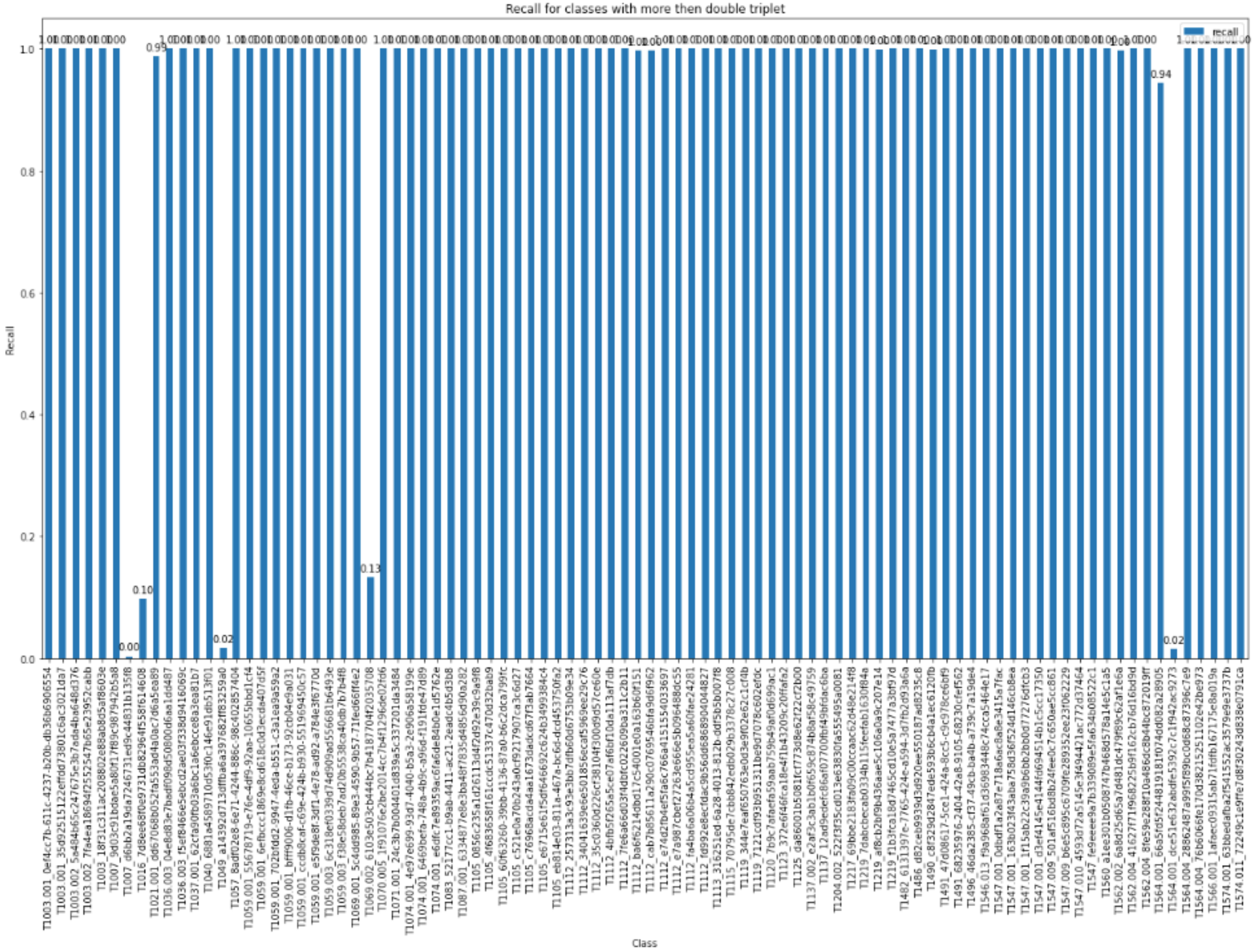


Difficulty

Prediction of more triplet case:



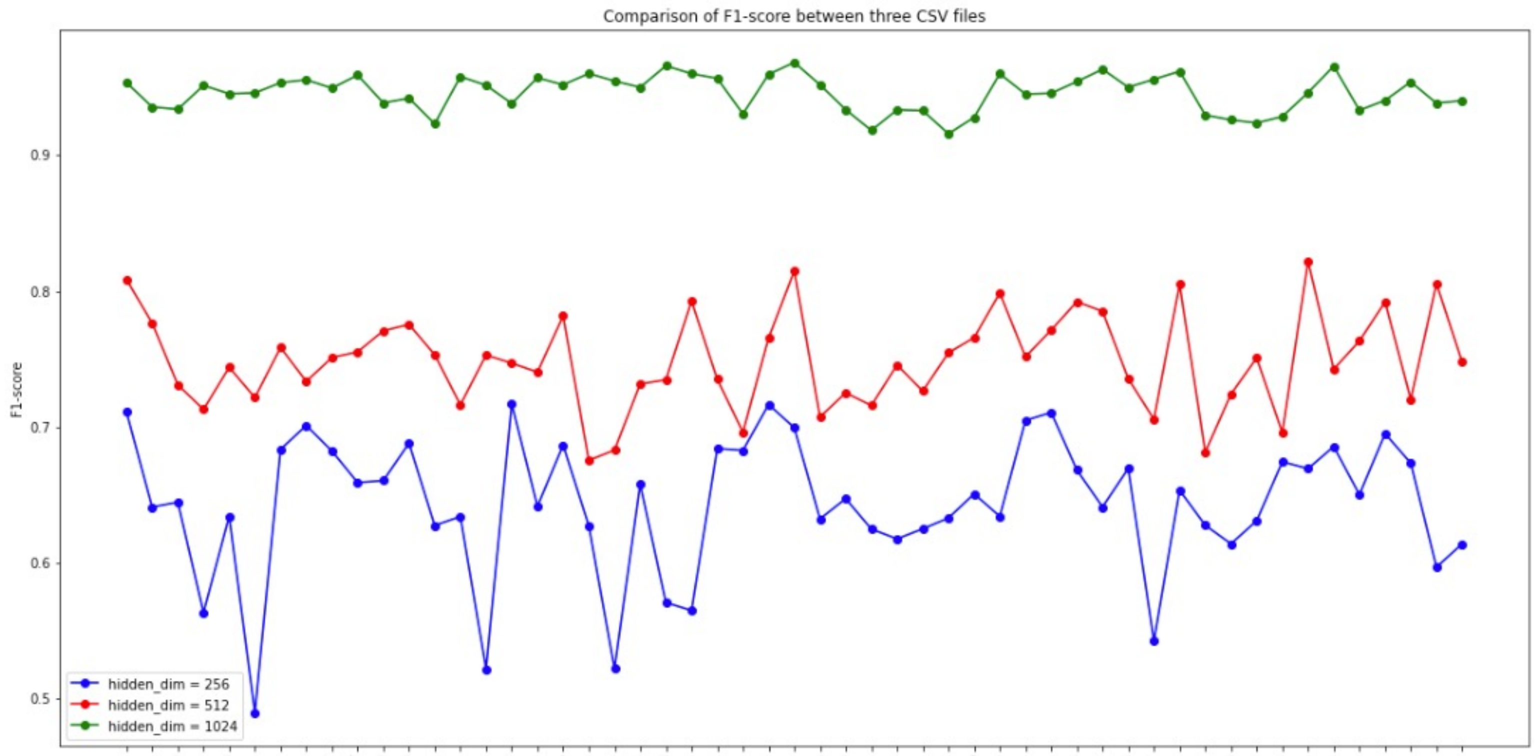
Number of rows: 90



Experiments

Oversampling

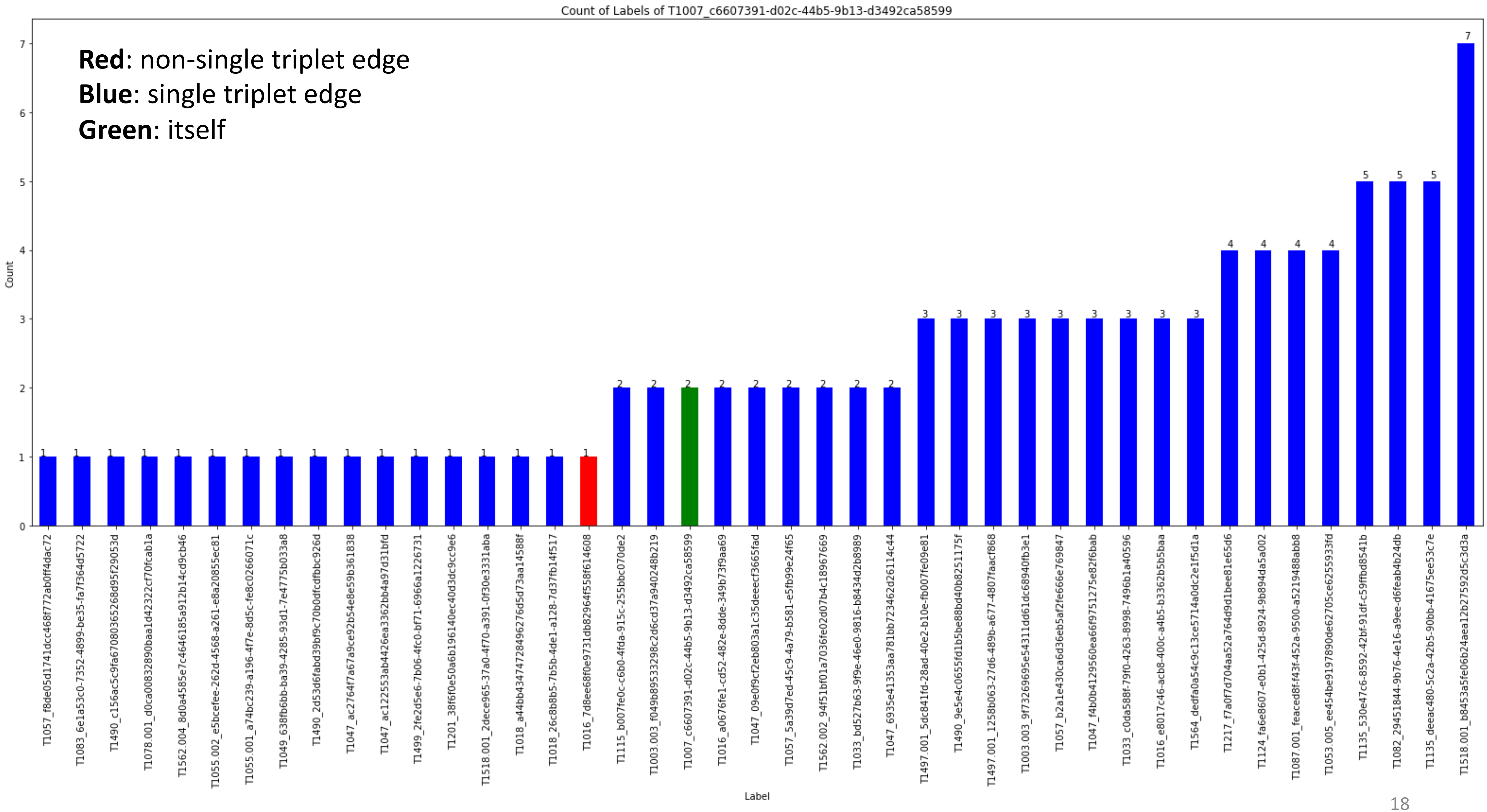
- Previous Trial: hope to see the result on **training** dataset
  - Use data with **320** times single triplet → # of training data = 13657600
  - Larger hidden **dimension** → more neurons to remember the data
  - Let the model **overfit** first → Succeed → But **not generalized** in the testing set
  - Haven't try the **GraphSMOTE** yet



Thought

- To observe the real distribution of the prediction
  - Figure out what the model really predict about
  - Do some **further experiments** based on the distribution of the prediction
    - Remove popular labels
    - Top3, Top5…
  - Figure out why the **T1518_c9b** is so well predicted

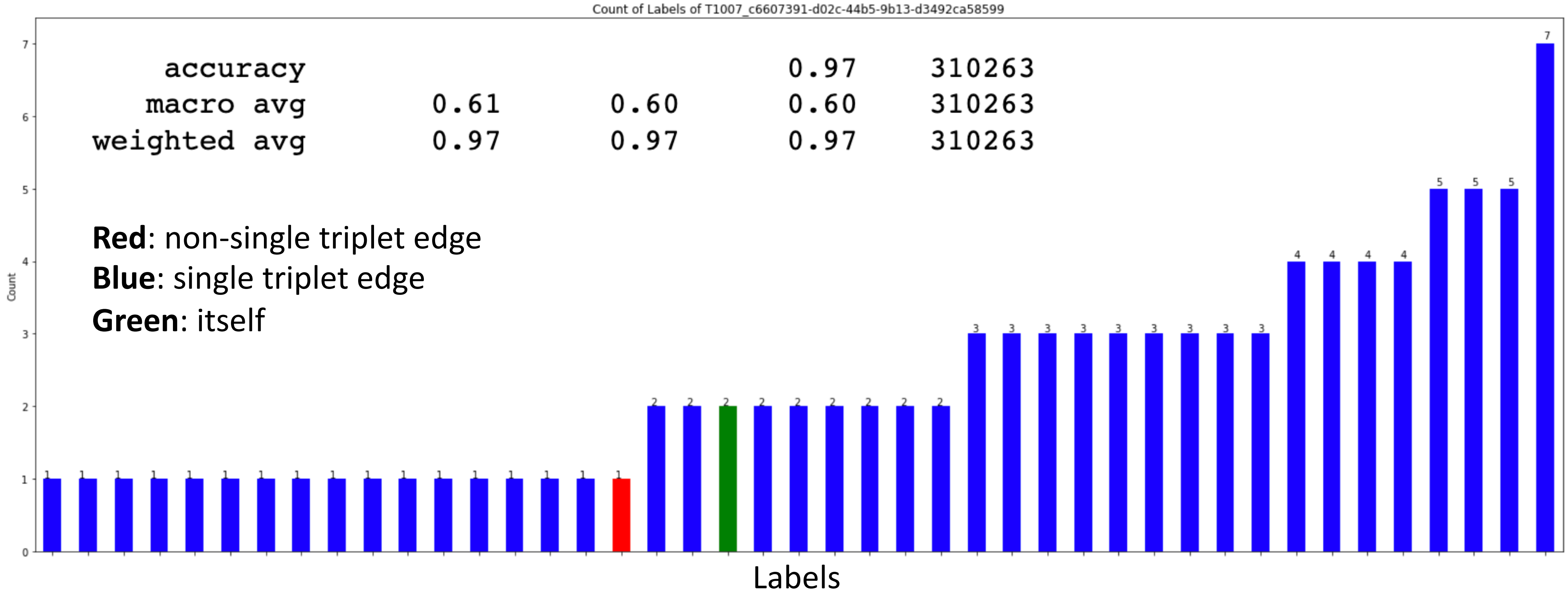
- So I plot the distribution of the prediction of the single triplet cases:


Count of Labels of T1007_c6607391-d02c-44b5-9b13-d3492ca58599
Red: non-single triplet edge
Blue: single triplet edge
Green: itself
Count
Label
T1057_f8de05d1741dcc468f772ab0ff4dac72
T1083_6e1a53c0-7352-4899-be35-fa7f364d5722
T1490_c156ac5c9fa67080365268d95f29053d
T1078.001_d0ca00832890baa1d42322cf70fcab1a
T1562.004_8d0a4585e7c4646185a912b14cd9cb46
T1055.002_e5bcefee-262d-4568-a261-e8a20855ec81
T1055.001_a74bc239-a196-4f7e-8d5c-fe8c0266071c
T1049_638fb6bb-ba39-4285-93d1-7e4775b033a8
T1490_2d53d6fabd39bf9c70b0dfcdfbbc926d
T1047_ac2764f7a67a9ce92b54e8e59b361838
T1047_ac122553ab4426ea3362bb4a97d31bfd
T1499_2fe2d5e6-7b06-4fc0-bf71-6966a1226731
T1201_38f6f0e50a6b196140ec40d3dc9cc9e6
T1518.001_2dece965-37a0-4f70-a391-0f30e3331aba
T1018_a44bb43474728496276d5d73aa14588f
T1018_26c8b8b5-7b5b-4de1-a128-7d37fb14f517
T1016_7d8ee68f0e9731db82964f558f614608
T1115_b007fe0c-c6b0-4fda-915c-255bbc070de2
T1003.003_f049b89533298c2d6cd37a940248b219
T1007_c6607391-d02c-44b5-9b13-d3492ca58599
T1016_a0676fe1-cd52-482e-8dde-349b73f9aa69
T1047_09e0f9cf2eb803a1c35deeecf3665fad
T1057_5a39d7ed-45c9-4a79-b581-e5fb99e24f65
T1562.002_94f51bf01a7036fe02d07b4c18967669
T1033_bd527b63-9f9e-46e0-9816-b8434d2b8989
T1047_6935e41353aa781bb723462d26114c44
T1497.001_5dc841fd-28ad-40e2-b10e-fb007fe09e81
T1490_9e5e4c0655fd1b5be88bd40b8251175f
T1497.001_1258b063-27d6-489b-a677-4807faacf868
T1003.003_9f73269695e54311dd61dc68940fb3e1
T1057_b2a1e430ca6d36eb5af2fe666e769847
T1047_f4b0b4129560ea66f9751275e82f6bab
T1033_c0da588f-79f0-4263-8998-7496b1a40596
T1016_e8017c46-acb8-400c-a4b5-b3362b5b5baa
T1564_dedfa0a54c9c13ce5714a0dc2e1f5d1a
T1217_f7a0f7d704aa52a764d9d1bee81e65d6
T1124_fa6e8607-e0b1-425d-8924-9b894da5a002
T1087.001_feaced8f-f43f-452a-9500-a5219488abb8
T1053.005_ee454be9197890de62705ce6255933fd
T1135_530e47c6-8592-42bf-91df-c59ffbd8541b
T1082_29451844-9b76-4e16-a9ee-d6feab4b24db
T1135_deeac480-5c2a-42b5-90bb-41675ee53c7e
T1518.001_b8453a5fe06b24aea12b27592d5c3d3a

Observation of the Prediction

- Use the original training set

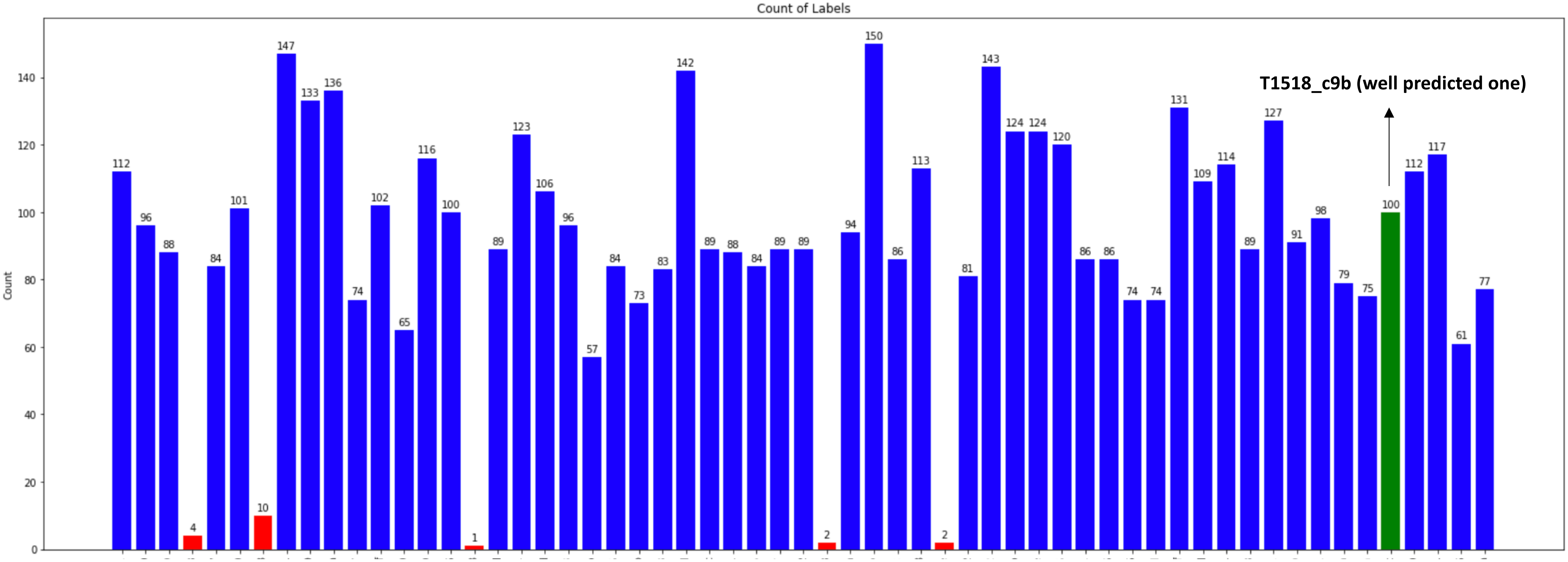
- The distribution of the prediction is so sparse
  - Most of the predictions are like this:



Observation of the Prediction

- The # of the predicted labels

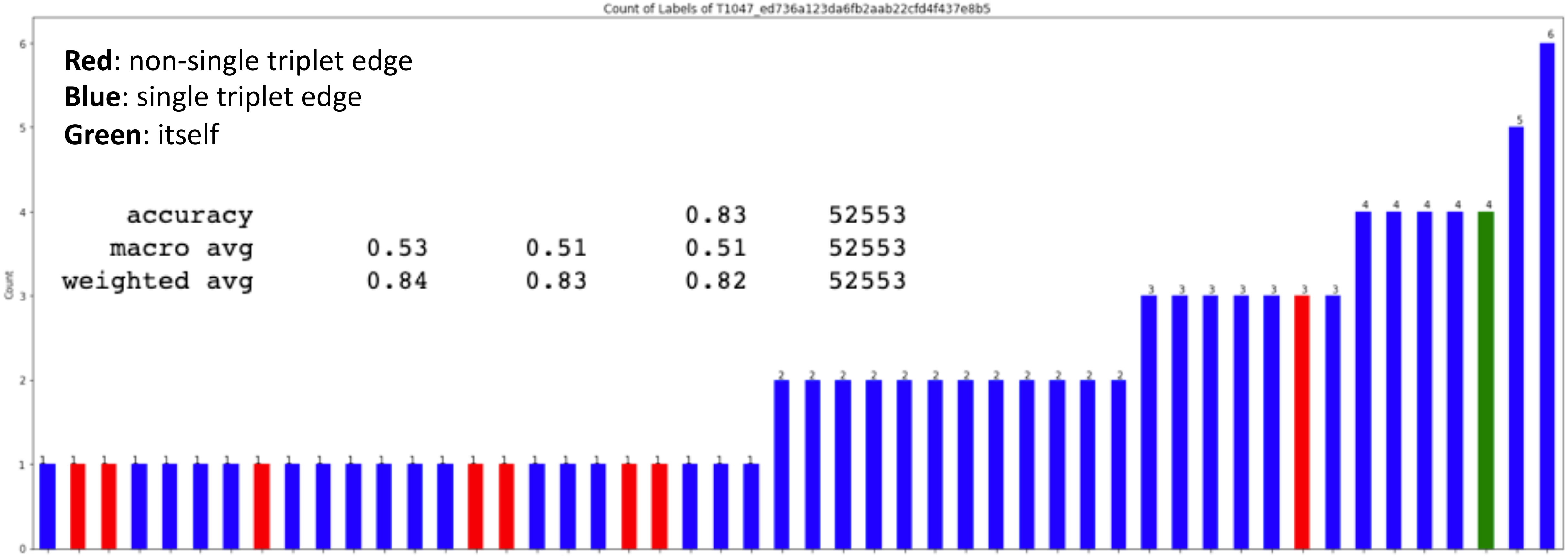
```
Top 5 Labels and Counts:
T1082_29451844-9b76-4e16-a9ee-d6feab4b24db: 150
T1016_921055f4-5970-4707-909e-62f594234d91: 147
T1124_fa6e8607-e0b1-425d-8924-9b894da5a002: 143
T1053.005_ee454be9197890de62705ce6255933fd: 142
T1016_e8017c46-acb8-400c-a4b5-b3362b5b5baa: 136
```



Experiment – Change the Dataset

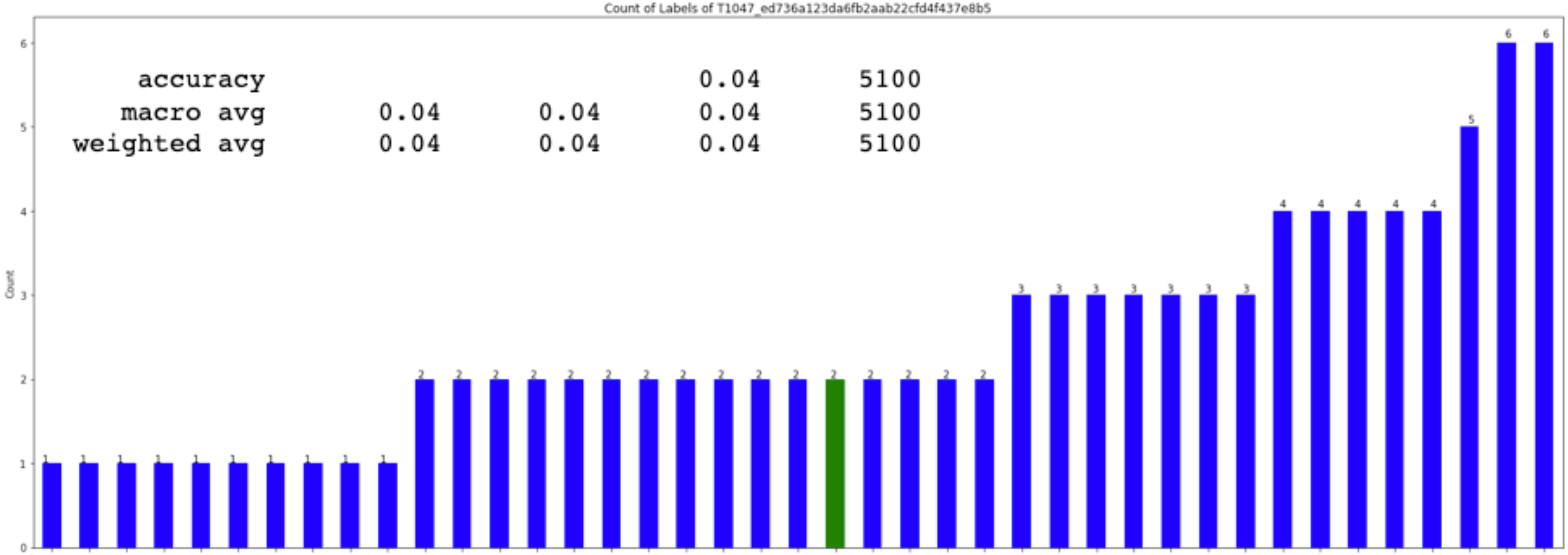
Observation of the Prediction

- Remove the **TTPs with support > 1000** (Removed 32 TTPs)
  - The predictions on the single triplet cases are still a mess



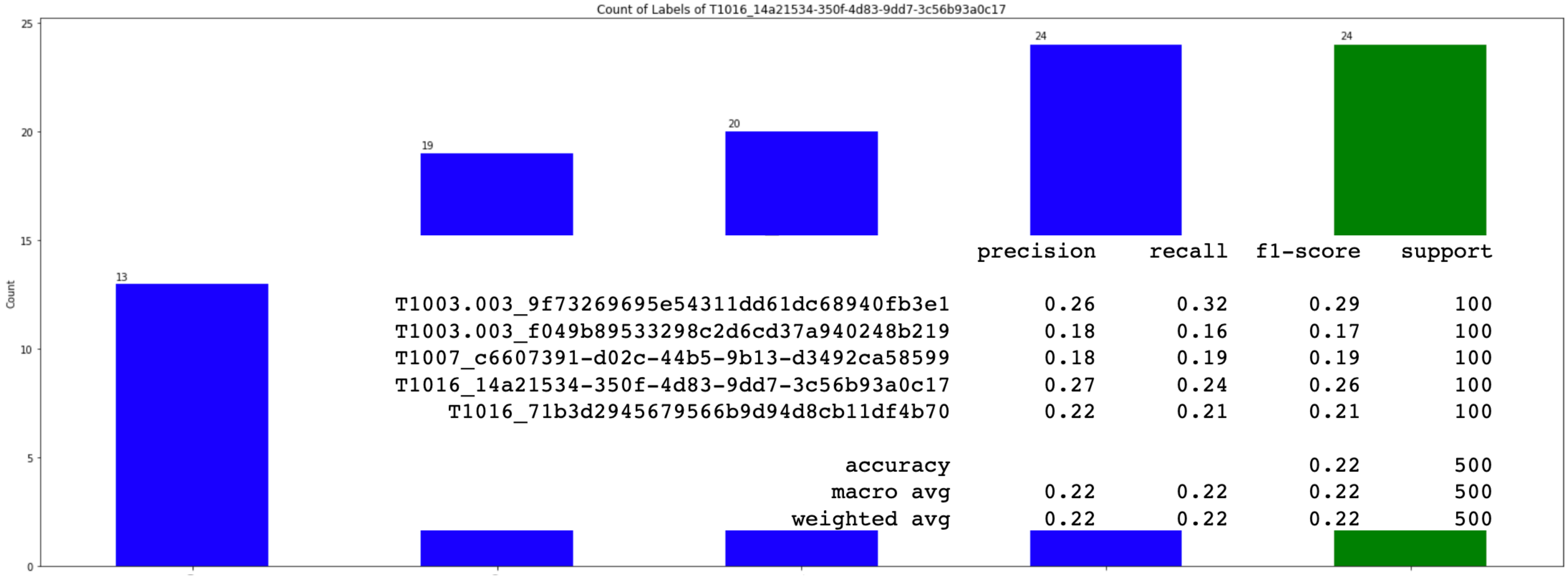
Observation of the Prediction

- Only use the **single triplet (support=100) cases**:
  - Still messy and even have a worse performance



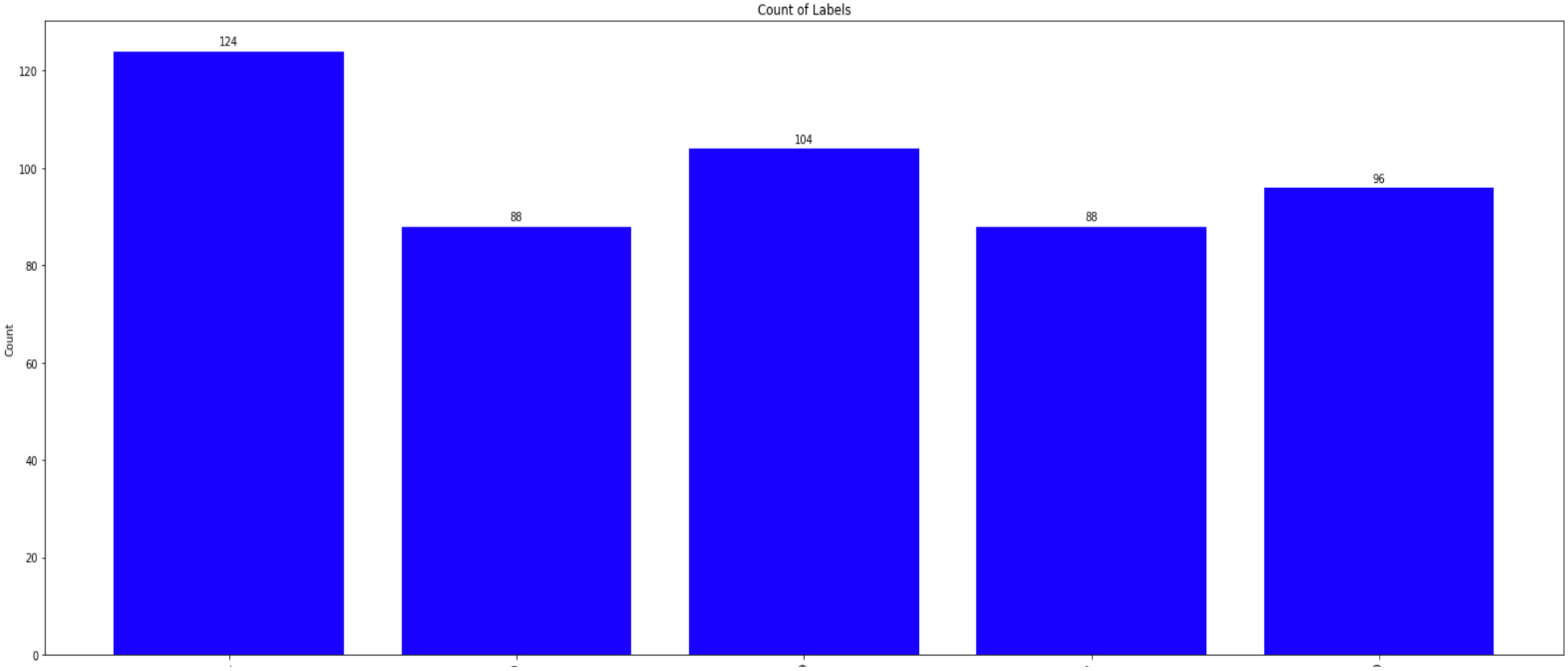
Observation of the Prediction

- Dataset only has the **5 single triplet labels**



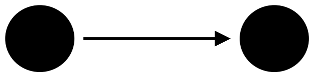
	precision	recall	f1-score	support
T1003.003_9f73269695e54311dd61dc68940fb3e1	0.26	0.32	0.29	100
T1003.003_f049b89533298c2d6cd37a940248b219	0.18	0.16	0.17	100
T1007_c6607391-d02c-44b5-9b13-d3492ca58599	0.18	0.19	0.19	100
T1016_14a21534-350f-4d83-9dd7-3c56b93a0c17	0.27	0.24	0.26	100
T1016_71b3d2945679566b9d94d8cb11df4b70	0.22	0.21	0.21	100
accuracy			0.22	500
macro avg	0.22	0.22	0.22	500
weighted avg	0.22	0.22	0.22	500

Observation of the Prediction



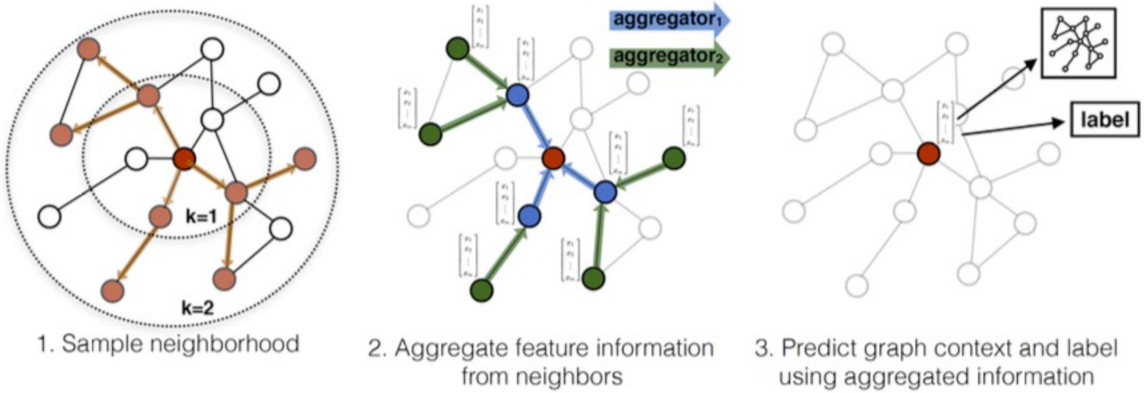
Conclusion

- GraphSAGE really have a bad performance on the **isolated** triplet case
  - Even training dataset consists of isolated triplet case only
  - Even training dataset consists of only 5 isolated triplet cases



- Possible Reason:
  - **Lack of Neighborhood Information**: GraphSAGE aggregates features from a node's local neighborhood to learn its representation. In the case of isolated triplets, each node has very limited neighborhood information (only one neighbor), which restricts the model's ability to learn complex or rich representations

Visual illustration of the GraphSAGE sample and aggregate approach:



Experiment – Predict More Labels

Predict the Top 3

- Thought:
  - The experts can use the DL prediction and combine with their expertise to make the final prediction
- Prediction Format:
  - Predict the **top 3** classes of the edge
  - True:

Row	Label
209570	125
209576	125
209579	125
209583	125
209594	125

Predicted:

Row	Labels		
209570	112	125	163
209576	124	158	125
209579	42	125	12
209583	112	125	33
209594	112	125	31

- Performance:

	precision	recall	f1-score
macro avg	0.656194	0.643040	0.634166
micro avg	0.977822	0.977822	0.977822

Previous one:

accuracy			0.97	310263
macro avg	0.61	0.60	0.60	310263
weighted avg	0.97	0.97	0.97	310263

Predict the Top 5

- Prediction Format:
  - Predict the **top 5** classes of the edge
  - True:

Row	Label
209569	125
209570	125
209576	125
209594	125
209605	125

Predicted:

Row	Labels				
209569	97	1	124	125	18
209570	112	125	163	53	42
209576	124	158	125	111	12
209594	157	125	31	112	33
209605	42	31	157	125	97

- Performance:

	precision	recall	f1-score
macro avg	0.701622	0.665422	0.656189
micro avg	0.979798	0.979798	0.979798

Previous one:

accuracy			0.97	310263
macro avg	0.61	0.60	0.60	310263
weighted avg	0.97	0.97	0.97	310263

Version of top3:

	precision	recall	f1-score
macro avg	0.656194	0.643040	0.634166
micro avg	0.977822	0.977822	0.977822

Experiment – DARPA

Change the DARPA

- Original Format:
  - {"subj": {"uuid": "F9F5DC58-9431-4519-82CF-2BBFF40796E9", "type": "Subject", "n_attrbiute": {"cmdline": "None", "type": "SUBJECT_THREAD", "pid": 8920}}, "relation": "EVENT_WRITE", "obj": {"uuid": "F5D43721-2AD3-487C-B3DA-0A5D551FAE0D", "type": "FileObject", "n_attrbiute": {"filepath": "\\\\Device\\\\HarddiskVolume2\\\\ProgramData\\\\Microsoft\\\\Windows Defender\\\\Network Inspection System\\\\Support\\\\NisLog.txt"}}, "timestamp": 1523295276588000000, "label": "benign"}

- Final Format:
  - Encode the subj uuid, relation, obj uuid and combine them with the label
  - Keep the mapping information into 2 mapping file
  - a is attack; b is benign

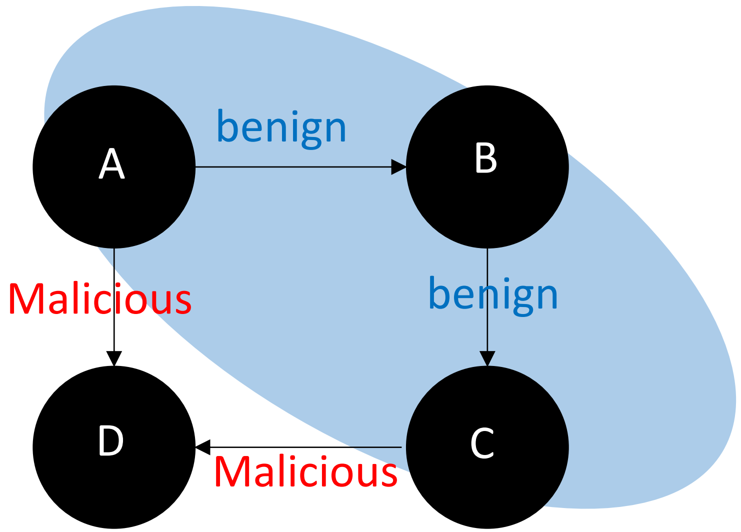
src	dest	rel	label
872529	1086092	5	a
872529	61317	7	a
1059095	759218	19	b
280074	759218	19	b

- Successfully run the code → preprocessing the data now → each json file needs **6 hours** (we have 21)

Future Work

Future Work

- Want the prediction like:
  - Not just predict the triplet
  - Predict a **series of triplets**
    - **Graph classification**?
    - Dataset?



Benign → for these 2 triplets

Malicious → for the whole graph

- Read the **GraphSAGE paper** and compare it to the GCN, GAT
- Figure out why the model can detect the **T1518_c9b** (if available?)
  - Is the label overlap or not overlap at all with other labels
- Figure out what the model really predict about
  - Observe the real data in the original form (before embedding) → overlapping?
- Use the Trans Family to get the embedding of the **DARPA** dataset's nodes and edges

Thanks!!

Appendix

Future Work

- Run the training of the DAPRA dataset
- Read the GraphSAGE paper and present it

- Figure out why the model can detect the **T1518_c9b** (if available?)
  - Is the label overlap or not overlap at all with other labels
- Figure out what the model really predict about
  - Observe the real data in the original form (before embedding) → overlapping?

GraphSMOTE: Imbalanced Node Classification on Graphs with Graph Neural Networks

WSDM '21, March 8–12, 2021, Virtual Event, Israel

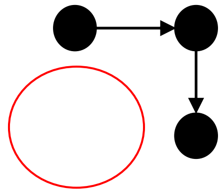
Tianxiang Zhao, Xiang Zhang, Suhang Wang

{tkz5084,xzz89,szw494}@psu.edu

College of Information Science and Technology, Penn State University State College, The USA

https://github.com/TianxiangZhao/GraphSmote

Model Design



```
g.ndata['feat'] = th.tensor(data["node_feat"])
g.edata['feat'] = th.tensor(data["edge_attr"])
g.edata['label'] = th.tensor(data["labels"])
```

- Concept from the DGL official website:
  1. Let the dgl graph's edge data have the attribute: **edata["label"]**
  2. Use **GraphSAGE** model to get the new **node embedding**
  3. Use **MLP** model to get the '**score**' of the **edge**
  4. **Concatenate** these two models
  5. Train the final model

For Node Embedding:

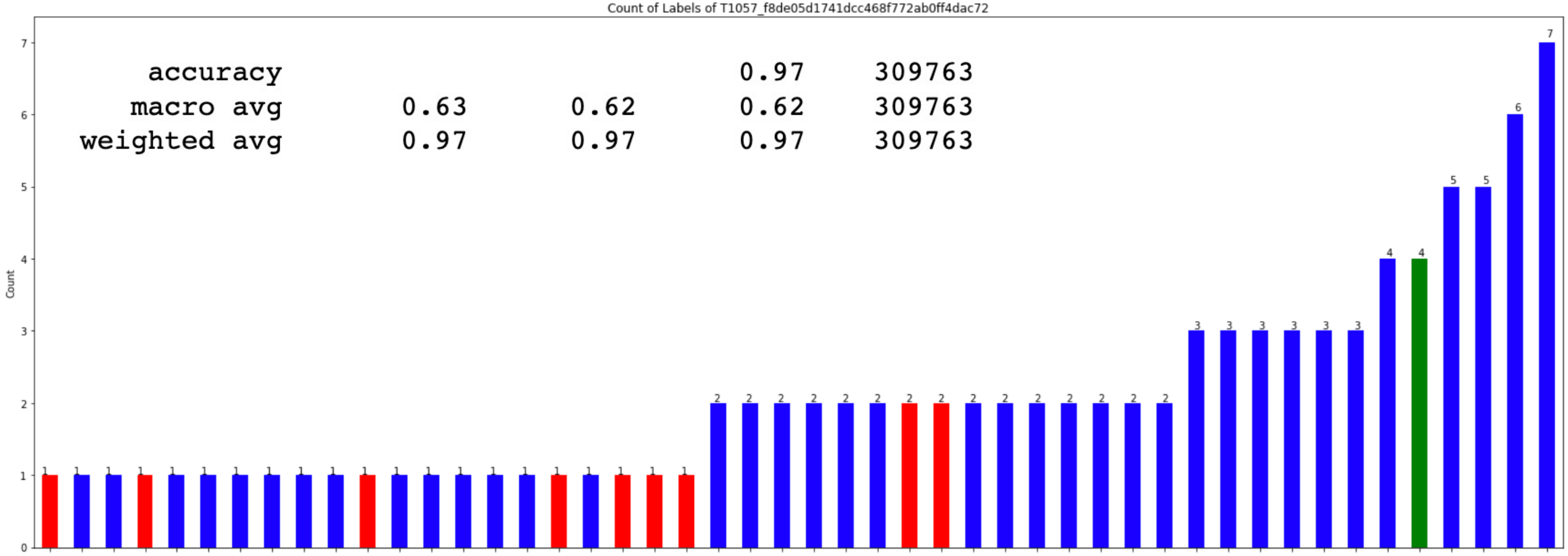
```
class GraphSAGE(nn.Module):
    def __init__(self, in_dim, hidden_dim, out_dim):
        super(GraphSAGE, self).__init__()
        self.layer1 = dglnn.SAGEConv(in_dim, hidden_dim, 'pool')
        self.layer2 = dglnn.SAGEConv(hidden_dim, out_dim, 'pool')

    def forward(self, g, inputs):
        h = self.layer1(g, inputs)
        h = torch.relu(h)
        h = self.layer2(g, h)
        return h
```

```
class MLPPredictor(nn.Module):
    def __init__(self, out_feats, out_classes):
        super().__init__()
        self.W = nn.Linear(out_feats*2, out_classes)

    def apply_edges(self, edges):
        h_u = edges.src['h']
        h_v = edges.dst['h']
        score = self.W(torch.cat([h_u, h_v], 1))
        return {'score': score}

    def forward(self, graph, h):
        with graph.local_scope():
            graph.ndata['h'] = h
            graph.apply_edges(self.apply_edges)
            return graph.edata['score']
```

Remove the Popular TTPs

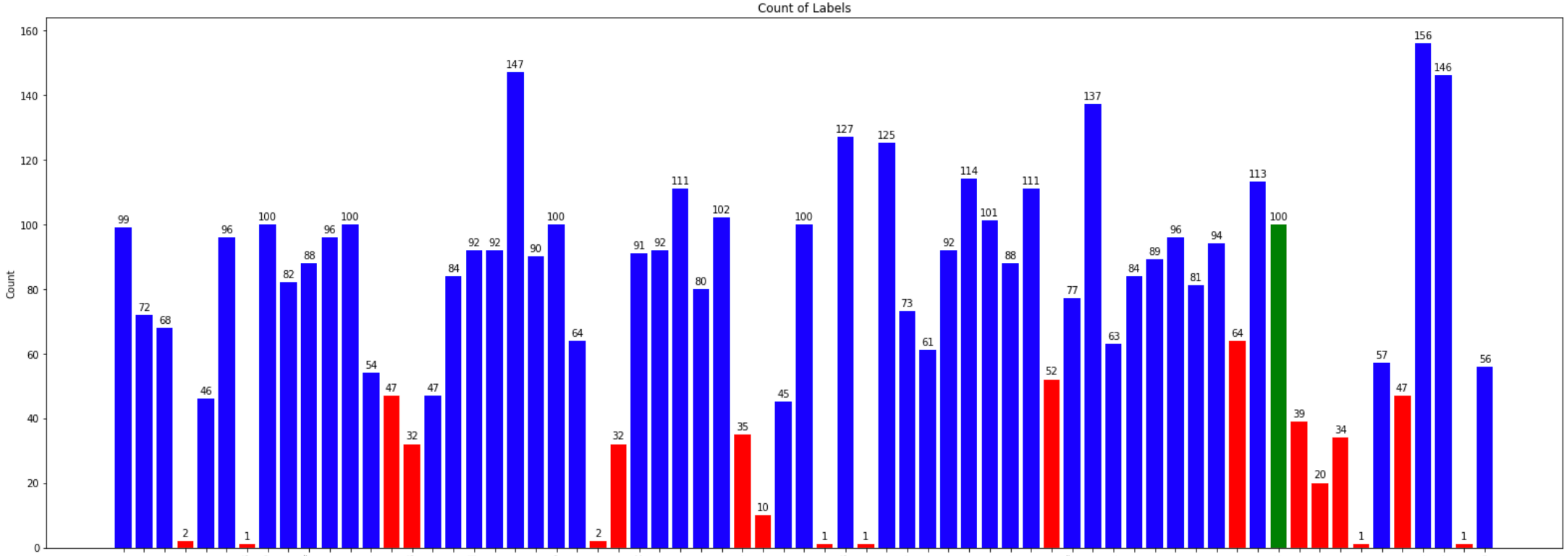
```
Top 5 Labels and Counts:
T1082_29451844-9b76-4e16-a9ee-d6feab4b24db: 150
T1016_921055f4-5970-4707-909e-62f594234d91: 147
T1124_fa6e8607-e0b1-425d-8924-9b894da5a002: 143
T1053.005_ee454be9197890de62705ce6255933fd: 142
T1016_e8017c46-acb8-400c-a4b5-b3362b5b5baa: 136
```

- Remove these 5 TTPs from the dataset and then train it again:
  - More prediction on the non-single triplet case is like this:

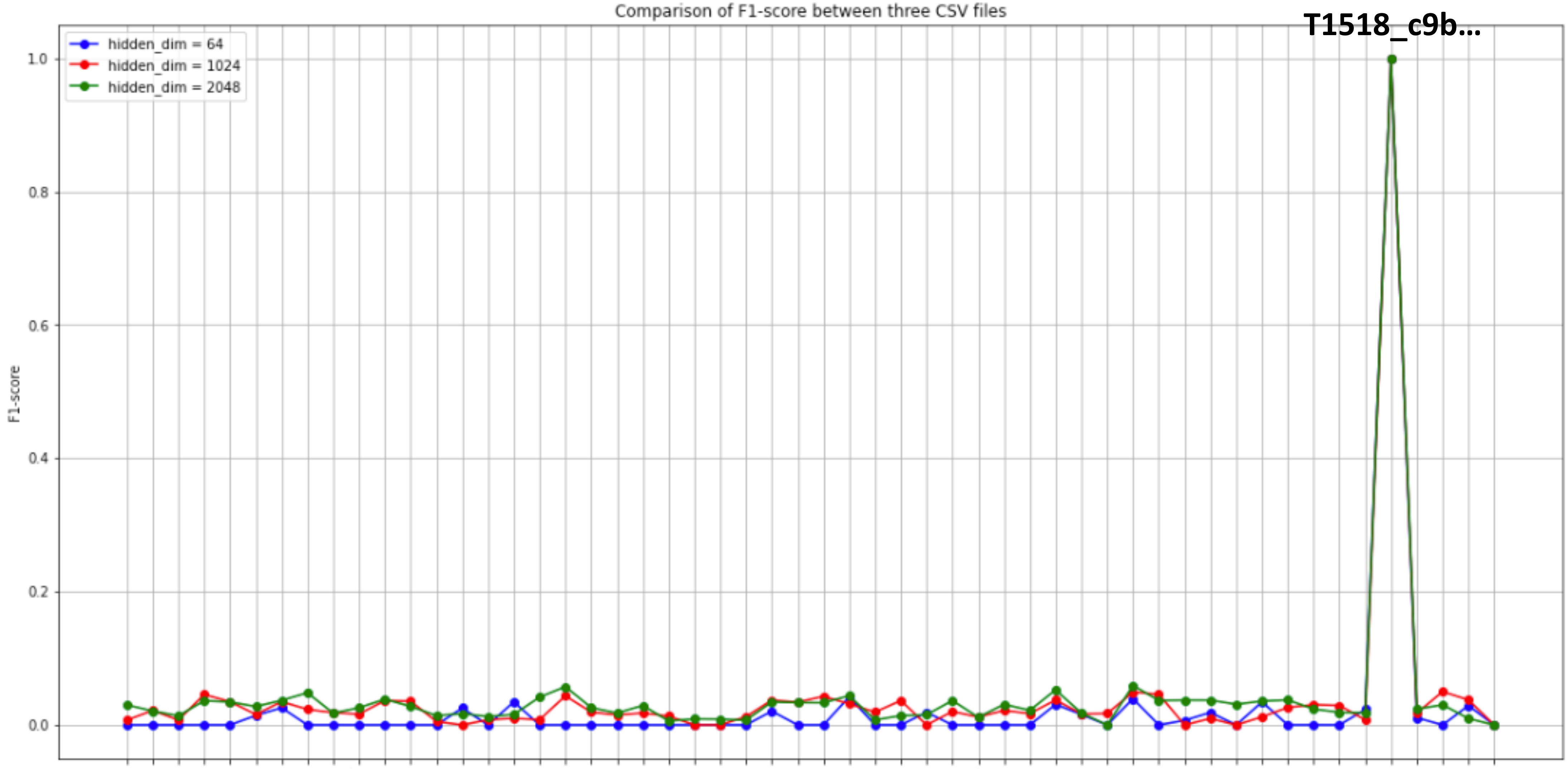


Remove the Popular TTPs

- Remove these 5 TTPs from the dataset and then train it again:
- The # of the predicted labels



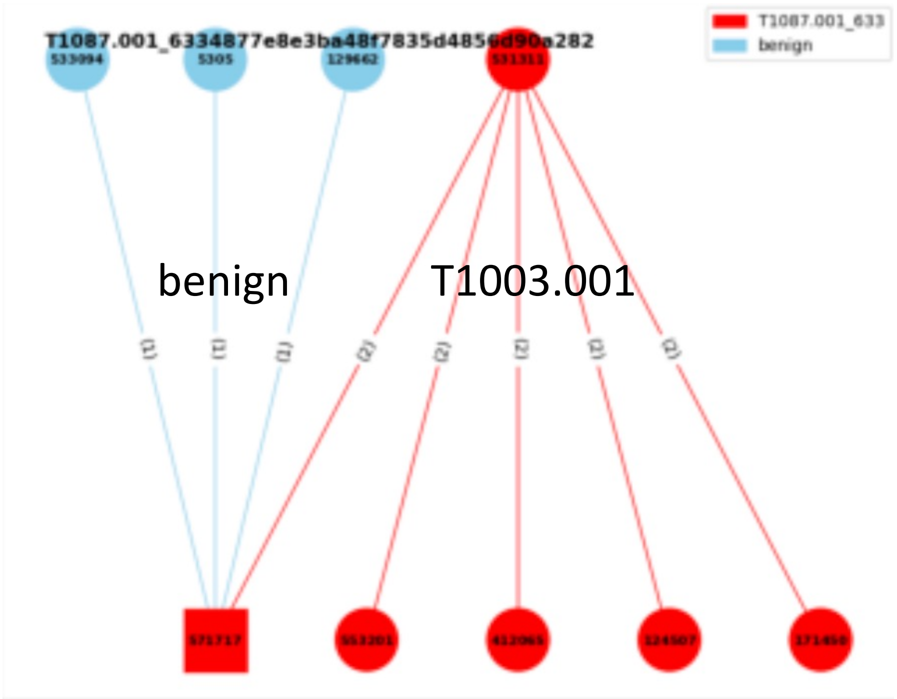
Observation on Different Dimension



Experiment 3

- **Experiment 3:**
  - Consider the **neighbor** benign nodes
  - Edge classification

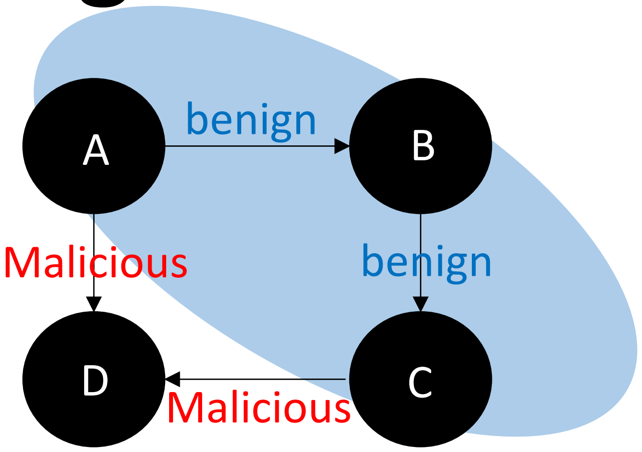
  - Given a graph → label the triplets with the benign or the specific AP



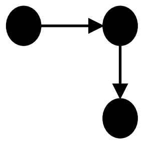
NOTE!!

NOTE of the meeting with Prof. Huang

- Want the prediction like:



Benign → for these 2 triplets



Malicious → for the whole graph

- Demonstrate how I do the classification clearly to the team
  - Input: whole graph
  - Prediction: Triplet by triplet

- Why GraphSAGE?
  - Compare to GCN, GAT
  - What's the mathematic implementation of the model → paper

NOTE from Prof. Sun

- 每次都要再複習一下問題點
  - Show 以前的數據等等
  - 為何要做現在這個實驗
- 圖表要清楚一點
  - X and Y-axis
- 給的example要是真的有解決那個問題的example