In [1]:
```
pip install torch_geometric
```

```
Requirement already satisfied: torch_geometric in c:\users\sahim\anaconda3
\lib\site-packages (2.5.2)
Requirement already satisfied: tqdm in c:\users\sahim\anaconda3\lib\site-p
ackages (from torch_geometric) (4.65.0)
Requirement already satisfied: numpy in c:\users\sahim\anaconda3\lib\site-
packages (from torch_geometric) (1.24.3)
Requirement already satisfied: scipy in c:\users\sahim\anaconda3\lib\site-
packages (from torch_geometric) (1.10.1)
Requirement already satisfied: fsspec in c:\users\sahim\anaconda3\lib\site
-packages (from torch_geometric) (2023.3.0)
Requirement already satisfied: jinja2 in c:\users\sahim\anaconda3\lib\site
-packages (from torch_geometric) (3.1.2)
Requirement already satisfied: aiohttp in c:\users\sahim\anaconda3\lib\sit
e-packages (from torch_geometric) (3.8.3)
Requirement already satisfied: requests in c:\users\sahim\anaconda3\lib\si
te-packages (from torch_geometric) (2.31.0)
Requirement already satisfied: pyparsing in c:\users\sahim\anaconda3\lib\s
ite-packages (from torch_geometric) (3.0.9)
Requirement already satisfied: scikit-learn in c:\users\sahim\anaconda3\li
b\site-packages (from torch_geometric) (1.3.0)
Requirement already satisfied: psutil>=5.8.0 in c:\users\sahim\anaconda3\l
ib\site-packages (from torch_geometric) (5.9.0)
Requirement already satisfied: attrs>=17.3.0 in c:\users\sahim\anaconda3\l
ib\site-packages (from aiohttp->torch_geometric) (22.1.0)
Requirement already satisfied: charset-normalizer<3.0,>=2.0 in c:\users\sa
him\anaconda3\lib\site-packages (from aiohttp->torch_geometric) (2.0.4)
Requirement already satisfied: multidict<7.0,>=4.5 in c:\users\sahim\anaco
nda3\lib\site-packages (from aiohttp->torch_geometric) (6.0.2)
Requirement already satisfied: async-timeout<5.0,>=4.0.0a3 in c:\users\sah
im\anaconda3\lib\site-packages (from aiohttp->torch_geometric) (4.0.2)
Requirement already satisfied: yarl<2.0,>=1.0 in c:\users\sahim\anaconda3
\lib\site-packages (from aiohttp->torch_geometric) (1.8.1)
Requirement already satisfied: frozenlist>=1.1.1 in c:\users\sahim\anacond
a3\lib\site-packages (from aiohttp->torch_geometric) (1.3.3)
Requirement already satisfied: aiosignal>=1.1.2 in c:\users\sahim\anaconda
3\lib\site-packages (from aiohttp->torch_geometric) (1.2.0)
Requirement already satisfied: MarkupSafe>=2.0 in c:\users\sahim\anaconda3
\lib\site-packages (from jinja2->torch_geometric) (2.1.1)
Requirement already satisfied: idna<4,>=2.5 in c:\users\sahim\anaconda3\li
b\site-packages (from requests->torch_geometric) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\users\sahim\anacon
da3\lib\site-packages (from requests->torch_geometric) (1.26.16)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\sahim\anacon
da3\lib\site-packages (from requests->torch_geometric) (2023.7.22)
Requirement already satisfied: joblib>=1.1.1 in c:\users\sahim\anaconda3\l
ib\site-packages (from scikit-learn->torch_geometric) (1.2.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\sahim\anac
onda3\lib\site-packages (from scikit-learn->torch_geometric) (2.2.0)
Requirement already satisfied: colorama in c:\users\sahim\anaconda3\lib\si
te-packages (from tqdm->torch_geometric) (0.4.6)
Note: you may need to restart the kernel to use updated packages.
```

In [2]:
```
%matplotlib inline

import numpy as np
import pandas as pd
import math
import json
import os
import collections

import matplotlib.pyplot as plt
import matplotlib
import matplotlib.animation as animation
import itertools
from itertools import combinations


import sklearn
from sklearn import preprocessing
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler

from sklearn import metrics

import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.autograd import Variable

import torch_geometric
from torch_geometric.data import Data
from torch_geometric.data import DataLoader as DataLoaderGraph
from torch_geometric.data import Dataset as DatasetGraph
from torch_geometric.data import Batch as BatchGraph

#from torch_geometric.transforms import AddTrainValTestMask as masking

from torch_geometric.nn import GCNConv, BatchNorm, SAGEConv, SGConv, ChebCo
from torch_geometric.utils.convert import to_networkx

import networkx as nx
```

In [3]:
```
with open("musae_git_features.json") as json_data:
    data_raw = json.load(json_data)

edges=pd.read_csv("musae_git_edges.csv")
target_df=pd.read_csv("musae_git_target.csv")#.to_numpy()[:,2]
```

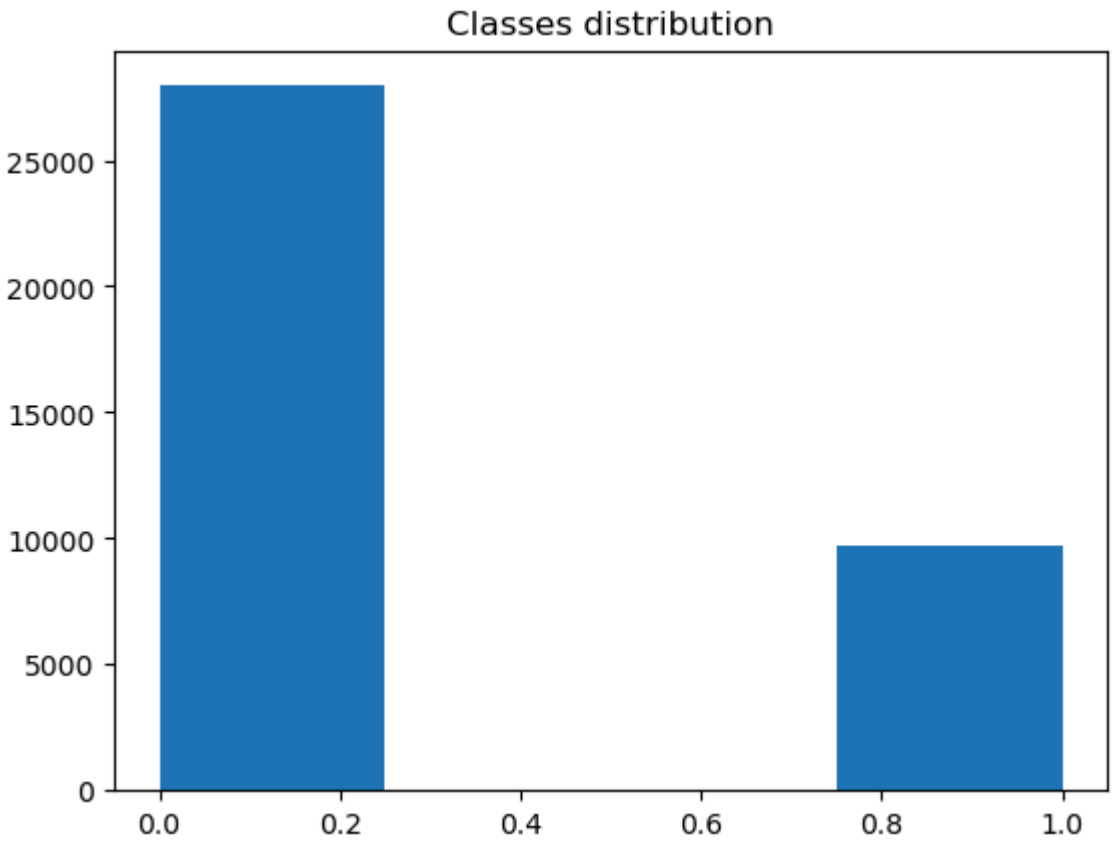
In [4]:
```
print("5 top nodes labels")
print(target_df.head(5).to_markdown())
print()
print("5 last nodes")
print(target_df.tail(5).to_markdown())
```

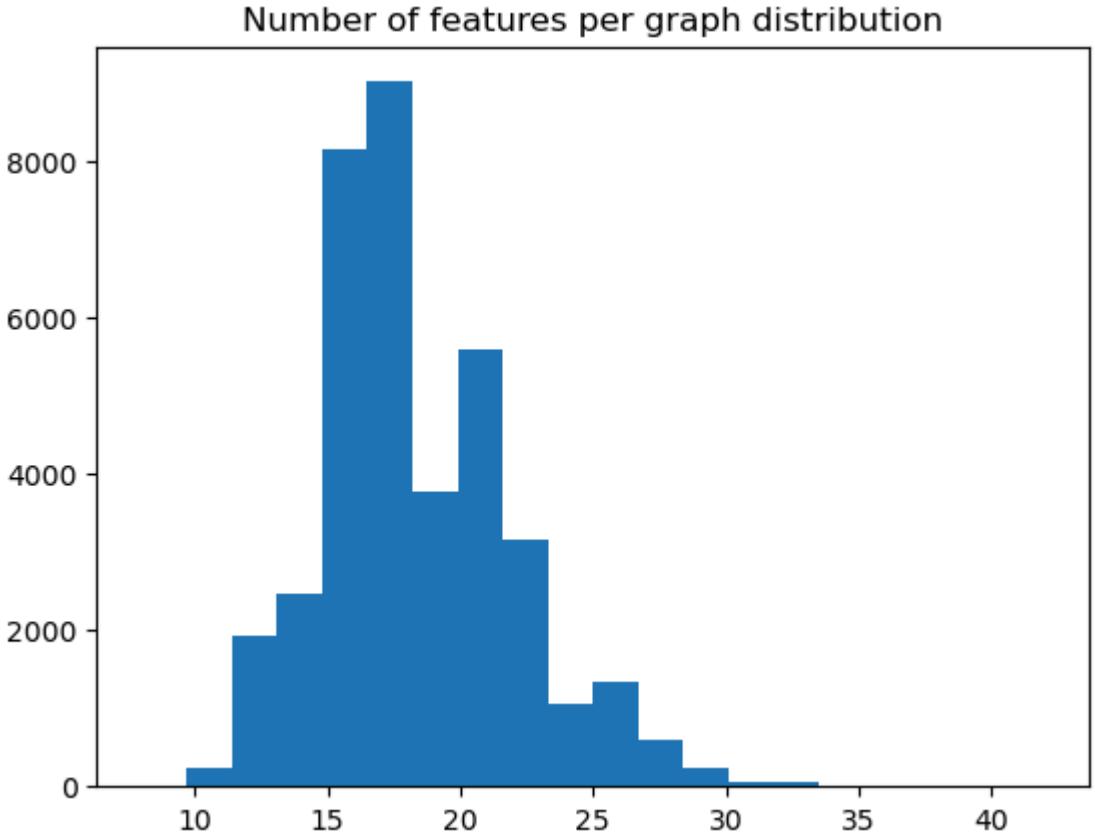
```
5 top nodes labels
|    |   id | name         |   ml_target |
|---:|-----:|:-------------|------------:|
|  0 |    0 | Eiryyy       |           0 |
|  1 |    1 | shawflying   |           0 |
|  2 |    2 | JpMCarrilho  |           1 |
|  3 |    3 | SuhwanCha    |           0 |
|  4 |    4 | sunilangadi2 |           1 |

5 last nodes
|       |    id | name           |   ml_target |
|------:|------:|:---------------|------------:|
| 37695 | 37695 | shawnwanderson |           1 |
| 37696 | 37696 | kris-ipeh      |           0 |
| 37697 | 37697 | qpautrat       |           0 |
| 37698 | 37698 | Injabie3       |           1 |
| 37699 | 37699 | caseycavanagh  |           0 |
```

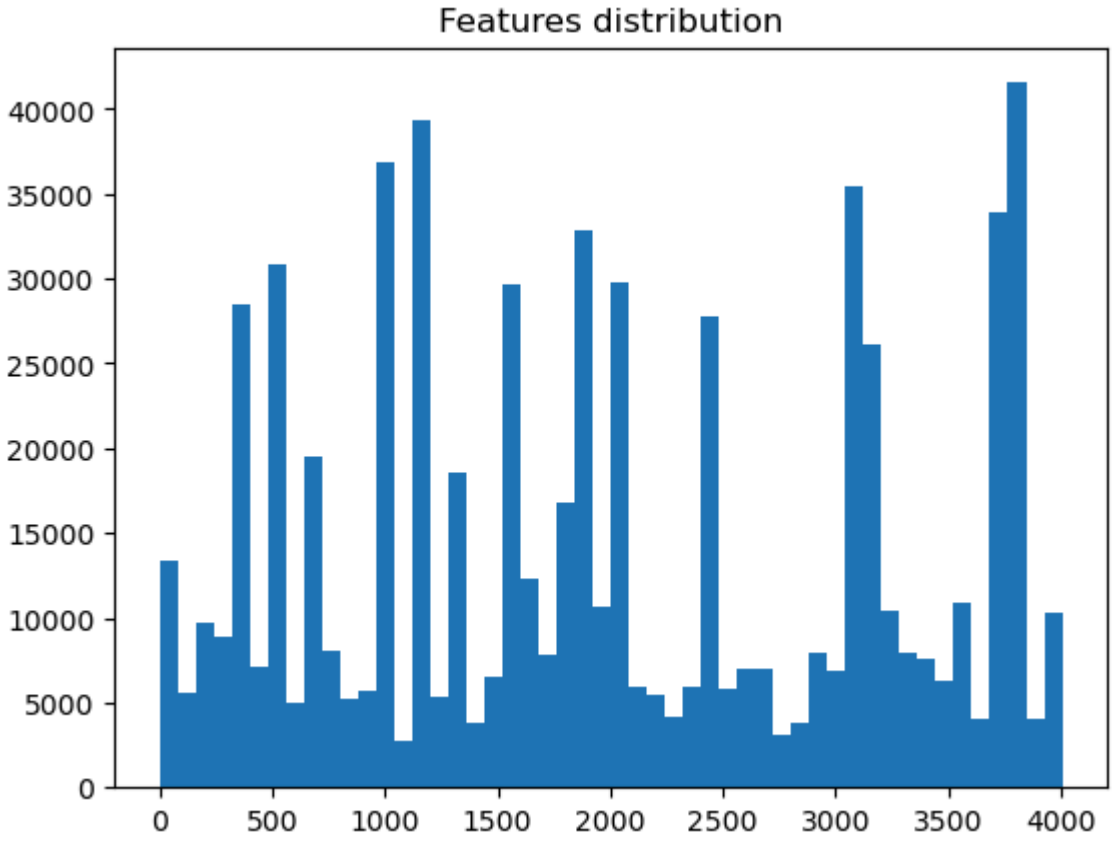
In [5]:
```
feats=[]
feat_counts=[]
for i in range(len(data_raw)):
    feat_counts+=[len(data_raw[str(i)])]
    feats+=data_raw[str(i)]

print("5 top nodes labels")
print(target_df.head(5).to_markdown())
print()
print("5 last nodes")
print(target_df.tail(5).to_markdown())
```

```
5 top nodes labels
|    |   id | name         |   ml_target |
|---:|-----:|:-------------|------------:|
|  0 |    0 | Eiryyy       |           0 |
|  1 |    1 | shawflying   |           0 |
|  2 |    2 | JpMCarrilho  |           1 |
|  3 |    3 | SuhwanCha    |           0 |
|  4 |    4 | sunilangadi2 |           1 |

5 last nodes
|       |    id | name           |   ml_target |
|------:|------:|:---------------|------------:|
| 37695 | 37695 | shawnwanderson |           1 |
| 37696 | 37696 | kris-ipeh      |           0 |
| 37697 | 37697 | qpautrat       |           0 |
| 37698 | 37698 | Injabie3       |           1 |
| 37699 | 37699 | caseycavanagh  |           0 |
```

In [6]:
```
plt.hist(target_df.ml_target,bins=4,);
plt.title("Classes distribution")
plt.show()

plt.hist(feat_counts,bins=20)
plt.title("Number of features per graph distribution")
plt.show()

plt.hist(feats,bins=50)
plt.title("Features distribution")
plt.show()
```




Number of features per graph distribution
8000
6000
4000
2000
0
10
15
20
25
30
35
40

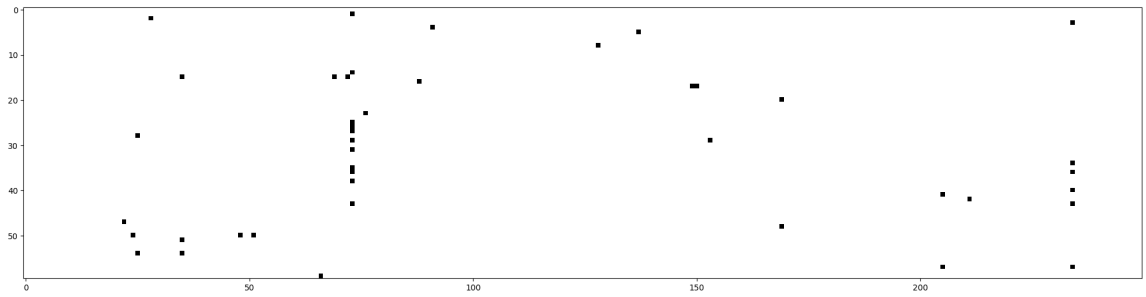

Features distribution
40000
35000
30000
25000
20000
15000
10000
5000
0
0
500
1000
1500
2000
2500
3000
3500
4000

In [7]:
```
counter=collections.Counter(feats)
print(list(counter.keys())[:10])
print(list(counter.values())[:10])
print(list(counter.keys())[-10:])
print(list(counter.values())[-10:])
#data_encoded
```

```
[1574, 3773, 3571, 2672, 2478, 2534, 3129, 3077, 1171, 2045]
[5045, 12726, 2486, 298, 165, 510, 22075, 28188, 24958, 21449]
[936, 172, 875, 3548, 2745, 793, 1941, 464, 2616, 3124]
[1, 2, 1, 2, 1, 1, 1, 1, 1, 1]
```

In [8]:
```
def encode_data(light=False,n=60):
    if light==True:
        nodes_included=n
    elif light==False:
        nodes_included=len(data_raw)

    data_encoded={}
    for i in range(nodes_included):#
        one_hot_feat=np.array([0]*(max(feats)+1))
        this_feat=data_raw[str(i)]
        one_hot_feat[this_feat]=1
        data_encoded[str(i)]=list(one_hot_feat)

    if light==True:
        sparse_feat_matrix=np.zeros((1,max(feats)+1))
        for j in range(nodes_included):
            temp=np.array(data_encoded[str(j)]).reshape(1,-1)
            sparse_feat_matrix=np.concatenate((sparse_feat_matrix,temp),axi
        sparse_feat_matrix=sparse_feat_matrix[1:,:]
        return(data_encoded,sparse_feat_matrix)
    elif light==False:
        return(data_encoded, None)
```

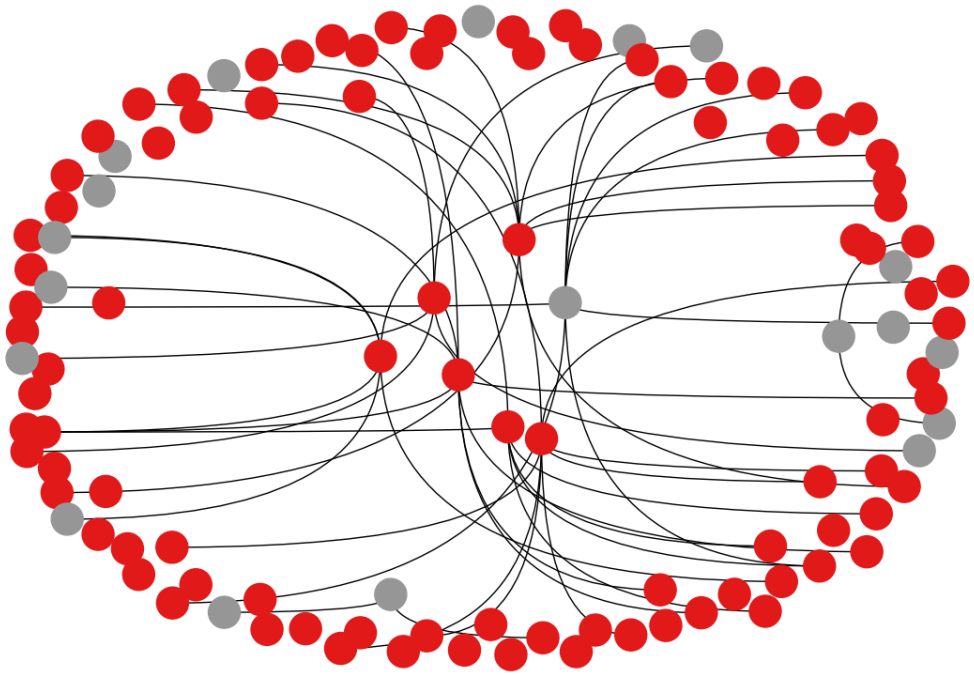
In [9]:
```
data_encoded_vis,sparse_feat_matrix_vis=encode_data(light=True,n=60)
plt.figure(figsize=(25,25));
plt.imshow(sparse_feat_matrix_vis[:,:250],cmap='Greys');
#plt.grid()
```



In [10]:
```
def construct_graph(data_encoded,light=False):
    node_features_list=list(data_encoded.values())
    node_features=torch.tensor(node_features_list)
    node_labels=torch.tensor(target_df['ml_target'].values)
    edges_list=edges.values.tolist()
    edge_index01=torch.tensor(edges_list, dtype = torch.long).T
    edge_index02=torch.zeros(edge_index01.shape, dtype = torch.long)#.T
    edge_index02[0,:]=edge_index01[1,:]
    edge_index02[1,:]=edge_index01[0,:]
    edge_index0=torch.cat((edge_index01,edge_index02),axis=1)
    g = Data(x=node_features, y=node_labels, edge_index=edge_index0)
    g_light = Data(x=node_features[:,0:2],
                   y=node_labels   ,
                   edge_index=edge_index0[:,:55])
    if light:
        return(g_light)
    else:
        return(g)
```

In [11]:
```
#g_top_draw = Data(x=node_features, y=node_labels, edge_index=edge_index0)

def draw_graph(data0):
    #node_labels=data0.y
    if data0.num_nodes>100:
        print("This is a big graph, can not plot...")
        return

    else:
        data_nx = to_networkx(data0)
        node_colors=data0.y[list(data_nx.nodes)]
        pos= nx.spring_layout(data_nx,scale =1)
        plt.figure(figsize=(12,8))
        nx.draw(data_nx, pos,  cmap=plt.get_cmap('Set1'),
                node_color =node_colors,node_size=600,connectionstyle="angl
                width =1, with_labels = False, edge_color = 'k', arrowstyle
```

In [12]:
```
g_sample=construct_graph(data_encoded=data_encoded_vis,light=True)
draw_graph(g_sample)
```



In [ ]:
```
data_encoded=encode_data(light=False)
```

In [ ]:
```
g=construct_graph(data_encoded=data_encoded,light=False)
```

In [ ]:
```
print(g)
```

In [ ]:
```
import torch_geometric.transforms as T
```

In [ ]:
```
import torch_geometric.transforms as T
msk=T.RandomNodeSplit(split="train_rest", num_splits = 1, num_val = 0.3, nu
g=msk(g)
print(g)
print()
print("training samples",torch.sum(g.train_mask).item())
print("validation samples",torch.sum(g.val_mask ).item())
print("test samples",torch.sum(g.test_mask ).item())
```

In [ ]:
```
class SocialGNN(torch.nn.Module):
    def __init__(self,num_of_feat,f):
        super(SocialGNN, self).__init__()


        self.conv1 = GCNConv(num_of_feat, f)

        self.conv2 = GCNConv(f, 2)


    def forward(self, data):
        x = data.x.float()
        edge_index =  data.edge_index

        x = self.conv1(x=x, edge_index=edge_index)
        x = F.relu(x)

        x = self.conv2(x, edge_index)
        return x
```

In [ ]:
```
def masked_loss(predictions,labels,mask):
    mask=mask.float()
    mask=mask/torch.mean(mask)
    loss=criterion(predictions,labels)
    loss=loss*mask
    loss=torch.mean(loss)
    return (loss)
```

In [ ]:
```
def masked_accuracy(predictions,labels,mask):
    mask=mask.float()
    mask/=torch.mean(mask)
    accuracy=(torch.argmax(predictions,axis=1)==labels).long()
    accuracy=mask*accuracy
    accuracy=torch.mean(accuracy)
    return (accuracy)
```

In [ ]:

```
def train_social(net,data,epochs=10,lr=0.01):
    optimizer = torch.optim.Adam(net.parameters(), lr=lr) # 00001
    best_accuracy=0.0

    train_losses=[]
    train_accuracies=[]

    val_losses=[]
    val_accuracies=[]

    test_losses=[]
    test_accuracies=[]

    for ep in range(epochs+1):
        optimizer.zero_grad()
        out=net(data)
        loss=masked_loss(predictions=out,
                         labels=data.y,
                         mask=data.train_mask)
        loss.backward()
        optimizer.step()
        train_losses+=[loss]
        train_accuracy=masked_accuracy(predictions=out,
                                       labels=data.y,
                                       mask=data.train_mask)
        train_accuracies+=[train_accuracy]

        val_loss=masked_loss(predictions=out,
                             labels=data.y,
                             mask=data.val_mask)
        val_losses+=[val_loss]

        val_accuracy=masked_accuracy(predictions=out,
                                     labels=data.y,
                                     mask=data.val_mask)
        val_accuracies+=[val_accuracy]

        test_accuracy=masked_accuracy(predictions=out,
                                      labels=data.y,
                                      mask=data.test_mask)
        test_accuracies+=[test_accuracy]
        if np.round(val_accuracy,4)> np.round(best_accuracy ,4):
            print("Epoch {}/{}, Train_Loss: {:.4f}, Train_Accuracy: {:.4f},
                      .format(ep+1,epochs, loss.item(), train_accuracy, val
            best_accuracy=val_accuracy




    plt.plot(train_accuracies)
    plt.plot(val_accuracies)
    plt.plot(test_accuracies)
    plt.show()

    print("Best accuracy",best_accuracy)
    return(best_accuracy)
```

In [ ]:
```
num_of_feat=g.num_node_features
net=SocialGNN(num_of_feat=num_of_feat,f=16)
criterion=nn.CrossEntropyLoss()
train_social(net,g,epochs=150,lr=0.1)
```