Fake News Detection with Pytorch Geometric

Installation

In [3]:
```
import torch
vers = torch.__version__
print("Torch vers: ", vers)

# PyG installation
!pip install -q torch-scatter -f https://pytorch-geometric.com/whl/torch-${TORCH}+${CUDA
!pip install -q torch-sparse -f https://pytorch-geometric.com/whl/torch-${TORCH}+${CUDA}
!pip install -q git+https://github.com/rusty1s/pytorch_geometric.git

import torch_geometric
```

```
Torch vers:  1.11.0+cu113
  Building wheel for torch-scatter (setup.py) ... done
     |████████████████████████████████| 48 kB 2.4 MB/s
  Building wheel for torch-sparse (setup.py) ... done
  Building wheel for torch-geometric (setup.py) ... done
```

Dataset

- Contains news propagation graphs extracted from Twitter with user profile data
- Source and raw data: https://github.com/KaiDMML/FakeNewsNet
- Reference: https://arxiv.org/pdf/2104.12259.pdf

In [33]:
```
from torch_geometric.datasets import UPFD
train_data = UPFD(root=".", name="gossipcop", feature="bert", split="train")
test_data = UPFD(root=".", name="gossipcop", feature="bert", split="test")
print("Train Samples: ", len(train_data))
print("Test Samples: ", len(test_data))
```

```
Train Samples:  1092
Test Samples:  3826
```

Investigating the News Propagation Graph

In [6]:
```
sample_id=1
train_data[sample_id].edge_index # shows the root nodes and corresponding connected node
```

Out[6]:
```
tensor([[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
           0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
           0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  15,  15,  17,  17,
          22,  22,  22,  22,  23,  26,  26,  26,  26,  26,  26,  26,  26,  27,
          27,  27,  28,  28,  28,  28,  28,  28,  28,  28,  28,  28,  28,  28,
          28,  28,  28,  28,  28,  28,  28,  28,  28,  28,  28,  28,  28,  28,
          28,  28,  28,  28,  28,  28,  28,  28,  30,  34,  34,  38,  38,  38,
          38,  39,  40,  57,  59,  65,  65,  65,  66,  68,  69,  77,  77,  79,
          79,  80,  83,  83,  83,  83,  84, 101, 110, 115, 116, 117],
        [  1,   2,   3,   4,   5,   6,   7,   8,   9,  10,  11,  12,  13,  14,
          15,  16,  17,  18,  19,  20,  21,  22,  23,  24,  25,  26,  27,  28,
          29,  30,  31,  32,  33,  34,  35,  36,  37,  38,  39,  40,  43,  44,
          45,  46,  47,  48,  49,  50,  51,  52,  53,  54,  55,  56,  57,  59,
          60,  61,  63,  64,  65,  66,  67,  68,  69,  70,  71,  72,  73,  74,
          75,  76,  77,  78,  79,  80,  81,  82,  83,  84,  85,  86,  87,  88,
          89,  90,  91,  92,  93,  94,  95,  96, 115, 119, 120, 121, 122, 123,
```

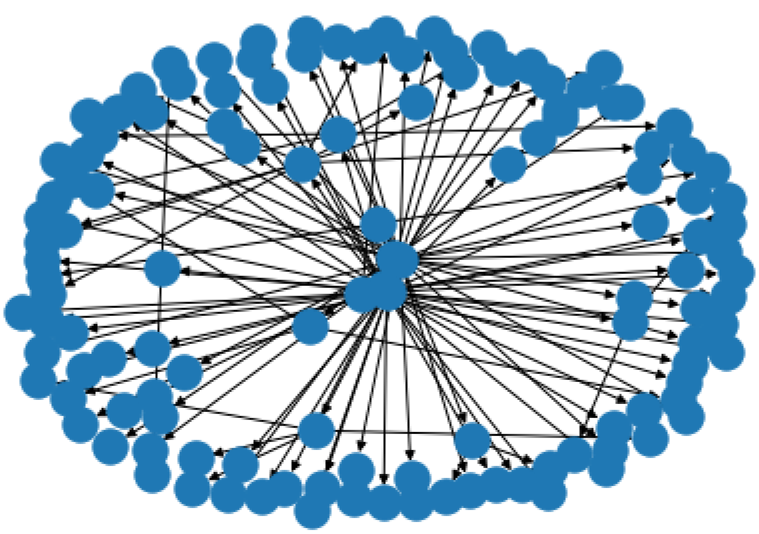
```
        124,   41,   42,   58,   62,   97,   98,   99, 100, 101, 103, 104, 105, 106,
        107, 108, 109, 110, 111, 112, 114, 102, 113, 116, 117, 118]])
```

In [7]:
```python
!pip install networkx
import networkx as nx

# From PyG utils
def to_networkx(data, node_attrs=None, edge_attrs=None, to_undirected=False,
                remove_self_loops=False):
    if to_undirected:
        G = nx.Graph()
    else:
        G = nx.DiGraph()
    G.add_nodes_from(range(data.num_nodes))
    node_attrs, edge_attrs = node_attrs or [], edge_attrs or []
    values = {}
    for key, item in data(*(node_attrs + edge_attrs)):
        if torch.is_tensor(item):
            values[key] = item.squeeze().tolist()
        else:
            values[key] = item
        if isinstance(values[key], (list, tuple)) and len(values[key]) == 1:
            values[key] = item[0]
    for i, (u, v) in enumerate(data.edge_index.t().tolist()):
        if to_undirected and v > u:
            continue
        if remove_self_loops and u == v:
            continue
        G.add_edge(u, v)
        for key in edge_attrs:
            G[u][v][key] = values[key][i]
    for key in node_attrs:
        for i, feat_dict in G.nodes(data=True):
            feat_dict.update({key: values[key][i]})
    return G
```

```
Requirement already satisfied: networkx in /usr/local/lib/python3.7/dist-packages (2.6.
3)
```

In [34]:
```python
nx.draw(to_networkx(train_data[sample_id]))
# nx.draw(to_networkx(train_data[2]))
```



Node features

In [10]:
```python
print(train_data[sample_id].x.shape)
```

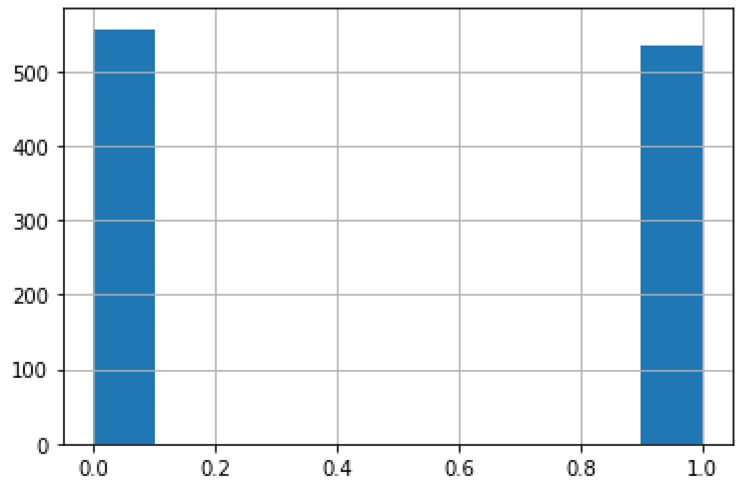
```
train_data[sample_id].x
```

Out[10]:
```
torch.Size([125, 768])
tensor([[-0.0895, -0.0141, -0.2686,  ...,  0.1738,  0.3480,  0.1831],
        [ 0.0600,  0.0440,  0.0087,  ...,  0.4195,  0.0239,  0.5018],
        [ 0.1362, -0.0738,  0.0080,  ...,  0.3674, -0.1551,  0.3255],
        ...,
        [-0.0284,  0.2227,  0.0766,  ...,  0.5024, -0.0791,  0.7929],
        [-0.0515,  0.1556,  0.2712,  ...,  0.4764,  0.0087,  0.4305],
        [ 0.0070,  0.1475,  0.0915,  ...,  0.5039, -0.0133,  0.6324]])
```

Class distribution

In [35]:
```
import pandas as pd
labels = [data.y.item() for i, data in enumerate(train_data)]
df = pd.DataFrame(labels, columns=["Labels"])
df["Labels"].hist()
```

Out[35]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f65030ba150>
```



Data Loaders

In [36]:
```
from torch_geometric.loader import DataLoader
train_loader = DataLoader(train_data, batch_size=128, shuffle=True)
test_loader = DataLoader(test_data, batch_size=128, shuffle=False)
```

Model and Training

--> Because it is a directed graph, it will only share information from the root

In [43]:
```
import torch
import torch.nn.functional as F
from torch.nn import BatchNorm1d, Linear, ReLU, Sequential

from torch_geometric.nn import GINConv, global_add_pool

class GINNet(torch.nn.Module):
    def __init__(self, in_channels, dim, out_channels):
        super().__init__()

        self.conv1 = GINConv(
            Sequential(Linear(in_channels, dim), BatchNorm1d(dim), ReLU(),
                       Linear(dim, dim), ReLU()))

        self.conv2 = GINConv(
```

```
            Sequential(Linear(dim, dim), BatchNorm1d(dim), ReLU(),
                       Linear(dim, dim), ReLU()))

        self.conv3 = GINConv(
            Sequential(Linear(dim, dim), BatchNorm1d(dim), ReLU(),
                       Linear(dim, dim), ReLU()))


        self.lin_news = Linear(in_channels, dim)

        self.lin1 = Linear(dim, dim)
        self.lin2 = Linear(2*dim, out_channels)

    def forward(self, x, edge_index, batch):
        h = self.conv1(x, edge_index)
        h = self.conv2(h, edge_index)
        h = self.conv3(h, edge_index)

        h = global_add_pool(h, batch)
        h = self.lin1(h).relu()
        h = F.dropout(h, p=0.5, training=self.training)
        #h = self.lin2(h)
        # According to UPFD paper: Include raw word2vec embeddings of news
        # This is done per graph in the batch
        root = (batch[1:] - batch[:-1]).nonzero(as_tuple=False).view(-1)
        root = torch.cat([root.new_zeros(1), root + 1], dim=0)
        # root is e.g. [   0,   14,   94,  171,  230,  302, ... ]
        news = x[root]
        news = self.lin_news(news).relu()

        out = self.lin2(torch.cat([h, news], dim=-1))
        return torch.sigmoid(out)

GINNet(train_data.num_features,128, 1)
```

Out[43]:

```
GINNet(
  (conv1): GINConv(nn=Sequential(
    (0): Linear(in_features=768, out_features=128, bias=True)
    (1): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=Tru
e)
    (2): ReLU()
    (3): Linear(in_features=128, out_features=128, bias=True)
    (4): ReLU()
  ))
  (conv2): GINConv(nn=Sequential(
    (0): Linear(in_features=128, out_features=128, bias=True)
    (1): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=Tru
e)
    (2): ReLU()
    (3): Linear(in_features=128, out_features=128, bias=True)
    (4): ReLU()
  ))
  (conv3): GINConv(nn=Sequential(
    (0): Linear(in_features=128, out_features=128, bias=True)
    (1): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=Tru
e)
    (2): ReLU()
    (3): Linear(in_features=128, out_features=128, bias=True)
    (4): ReLU()
  ))
  (lin_news): Linear(in_features=768, out_features=128, bias=True)
  (lin1): Linear(in_features=128, out_features=128, bias=True)
  (lin2): Linear(in_features=256, out_features=1, bias=True)
)
```

In [44]:

```
from sklearn.metrics import accuracy_score, f1_score
```

```python
device = torch.device('cpu')
model = GINNet(train_data.num_features,128,1).to(device)
optimizer = torch.optim.Adam(model.parameters(), lr=0.01)
loss_fnc = torch.nn.BCELoss()

def train(epoch):
    model.train()
    total_loss = 0
    for data in train_loader:
        data = data.to(device)
        optimizer.zero_grad()
        out = model(data.x, data.edge_index, data.batch)
        loss = loss_fnc(torch.reshape(out, (-1,)), data.y.float())
        loss.backward()
        optimizer.step()
        total_loss += float(loss) * data.num_graphs
    return total_loss / len(train_loader.dataset)


@torch.no_grad()
def test(epoch):
    model.eval()
    total_loss = 0
    all_preds = []
    all_labels = []
    for data in test_loader:
        data = data.to(device)
        out = model(data.x, data.edge_index, data.batch)
        loss = loss_fnc(torch.reshape(out, (-1,)), data.y.float())
        total_loss += float(loss) * data.num_graphs
        all_preds.append(torch.reshape(out, (-1,)))
        all_labels.append(data.y.float())

    # Calculate Metrics
    accuracy, f1 = metrics(all_preds, all_labels)

    return total_loss / len(test_loader.dataset), accuracy, f1


def metrics(preds, gts):
    preds = torch.round(torch.cat(preds))
    gts = torch.cat(gts)
    acc = accuracy_score(preds, gts)
    f1 = f1_score(preds, gts)
    return acc, f1
```

In [45]:

```python
for epoch in range(40):
    train_loss = train(epoch)
    test_loss, test_acc, test_f1 = test(epoch)
    print(f'Epoch: {epoch:02d} |  TrainLoss: {train_loss:.2f} | '
          f'TestLoss: {test_loss:.2f} | TestAcc: {test_acc:.2f} | TestF1: {test_f1:.2f}')
```

```
Epoch: 00 |  TrainLoss: 2.32 | TestLoss: 27.10 | TestAcc: 0.50 | TestF1: 0.00
Epoch: 01 |  TrainLoss: 0.50 | TestLoss: 3.50 | TestAcc: 0.61 | TestF1: 0.38
Epoch: 02 |  TrainLoss: 0.42 | TestLoss: 5.90 | TestAcc: 0.59 | TestF1: 0.31
Epoch: 03 |  TrainLoss: 0.38 | TestLoss: 2.11 | TestAcc: 0.74 | TestF1: 0.66
Epoch: 04 |  TrainLoss: 0.31 | TestLoss: 0.67 | TestAcc: 0.84 | TestF1: 0.83
Epoch: 05 |  TrainLoss: 0.30 | TestLoss: 0.45 | TestAcc: 0.85 | TestF1: 0.85
Epoch: 06 |  TrainLoss: 0.27 | TestLoss: 0.44 | TestAcc: 0.84 | TestF1: 0.85
Epoch: 07 |  TrainLoss: 0.25 | TestLoss: 0.62 | TestAcc: 0.84 | TestF1: 0.84
Epoch: 08 |  TrainLoss: 0.25 | TestLoss: 0.79 | TestAcc: 0.85 | TestF1: 0.84
Epoch: 09 |  TrainLoss: 0.19 | TestLoss: 0.62 | TestAcc: 0.84 | TestF1: 0.85
Epoch: 10 |  TrainLoss: 0.20 | TestLoss: 1.14 | TestAcc: 0.83 | TestF1: 0.83
Epoch: 11 |  TrainLoss: 0.22 | TestLoss: 0.44 | TestAcc: 0.85 | TestF1: 0.86
```

```
Epoch: 12 |  TrainLoss: 0.23 | TestLoss: 0.53 | TestAcc: 0.81 | TestF1: 0.84
Epoch: 13 |  TrainLoss: 0.25 | TestLoss: 0.45 | TestAcc: 0.83 | TestF1: 0.85
Epoch: 14 |  TrainLoss: 0.25 | TestLoss: 0.59 | TestAcc: 0.86 | TestF1: 0.85
Epoch: 15 |  TrainLoss: 0.17 | TestLoss: 0.80 | TestAcc: 0.86 | TestF1: 0.85
Epoch: 16 |  TrainLoss: 0.13 | TestLoss: 0.63 | TestAcc: 0.85 | TestF1: 0.86
Epoch: 17 |  TrainLoss: 0.12 | TestLoss: 0.61 | TestAcc: 0.85 | TestF1: 0.86
Epoch: 18 |  TrainLoss: 0.09 | TestLoss: 1.62 | TestAcc: 0.86 | TestF1: 0.85
Epoch: 19 |  TrainLoss: 0.06 | TestLoss: 0.84 | TestAcc: 0.90 | TestF1: 0.90
Epoch: 20 |  TrainLoss: 0.06 | TestLoss: 1.59 | TestAcc: 0.88 | TestF1: 0.87
Epoch: 21 |  TrainLoss: 0.06 | TestLoss: 0.62 | TestAcc: 0.88 | TestF1: 0.89
Epoch: 22 |  TrainLoss: 0.06 | TestLoss: 1.83 | TestAcc: 0.86 | TestF1: 0.85
Epoch: 23 |  TrainLoss: 0.05 | TestLoss: 2.74 | TestAcc: 0.88 | TestF1: 0.86
Epoch: 24 |  TrainLoss: 0.05 | TestLoss: 1.57 | TestAcc: 0.91 | TestF1: 0.90
Epoch: 25 |  TrainLoss: 0.04 | TestLoss: 1.16 | TestAcc: 0.91 | TestF1: 0.90
Epoch: 26 |  TrainLoss: 0.04 | TestLoss: 2.55 | TestAcc: 0.86 | TestF1: 0.84
Epoch: 27 |  TrainLoss: 0.04 | TestLoss: 1.51 | TestAcc: 0.91 | TestF1: 0.91
Epoch: 28 |  TrainLoss: 0.03 | TestLoss: 0.81 | TestAcc: 0.94 | TestF1: 0.94
Epoch: 29 |  TrainLoss: 0.02 | TestLoss: 1.28 | TestAcc: 0.91 | TestF1: 0.91
Epoch: 30 |  TrainLoss: 0.01 | TestLoss: 0.80 | TestAcc: 0.92 | TestF1: 0.92
Epoch: 31 |  TrainLoss: 0.01 | TestLoss: 0.89 | TestAcc: 0.92 | TestF1: 0.92
Epoch: 32 |  TrainLoss: 0.01 | TestLoss: 1.33 | TestAcc: 0.92 | TestF1: 0.92
Epoch: 33 |  TrainLoss: 0.01 | TestLoss: 1.37 | TestAcc: 0.92 | TestF1: 0.92
Epoch: 34 |  TrainLoss: 0.01 | TestLoss: 1.10 | TestAcc: 0.92 | TestF1: 0.92
Epoch: 35 |  TrainLoss: 0.01 | TestLoss: 1.33 | TestAcc: 0.90 | TestF1: 0.90
Epoch: 36 |  TrainLoss: 0.03 | TestLoss: 1.94 | TestAcc: 0.90 | TestF1: 0.90
Epoch: 37 |  TrainLoss: 0.03 | TestLoss: 1.43 | TestAcc: 0.91 | TestF1: 0.91
Epoch: 38 |  TrainLoss: 0.03 | TestLoss: 0.85 | TestAcc: 0.93 | TestF1: 0.93
Epoch: 39 |  TrainLoss: 0.04 | TestLoss: 0.49 | TestAcc: 0.91 | TestF1: 0.91
```

In [46]:

```
for data in test_loader:
    data = data.to(device)
    pred = model(data.x, data.edge_index, data.batch)
    df = pd.DataFrame()
    df["pred_logit"] = pred.detach().numpy()[:,0]
    df["pred"] = torch.round(pred).detach().numpy()[:,0]
    df["true"] = data.y.numpy()
    print(df.head(10))
    break
```

```
     pred_logit  pred  true
0  9.725100e-01   1.0     1
1  9.998444e-01   1.0     1
2  1.748089e-19   0.0     0
3  9.991243e-01   1.0     1
4  2.297928e-06   0.0     0
5  3.422373e-07   0.0     0
6  1.387274e-19   0.0     0
7  9.999346e-01   1.0     1
8  1.740764e-13   0.0     0
9  4.796627e-02   0.0     0
```

In [ ]: