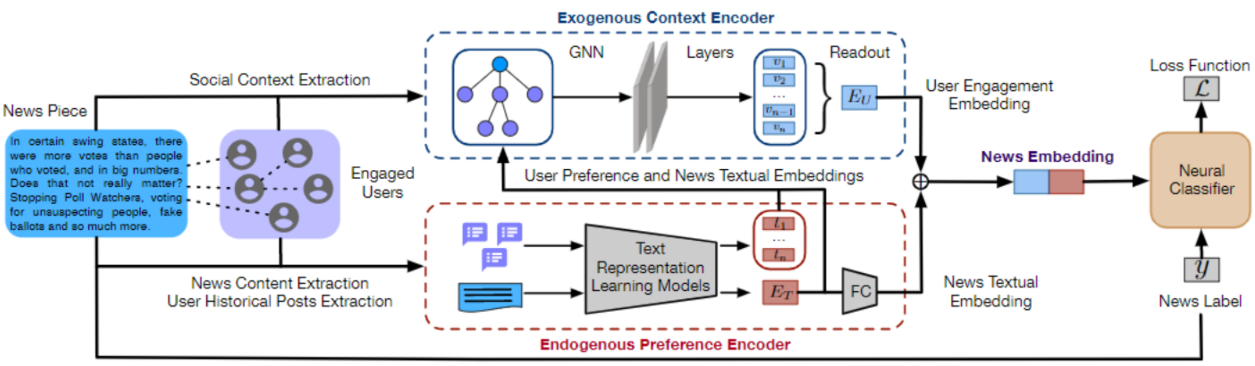
Graph propagation information과 BERT를 활용한 fake news detection model 구현

목차

1. Fake News Detection 개요

논문 출처: User Preference-aware Fake News Detection

Yingtong Dou1, Kai Shu2, Congying Xia1, Philip S. Yu1, Lichao Sun3. 2021. 'User Preference-aware Fake News Detection', ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '21), July 11–15, 2021, 5 pages.



1. 기존 대부분의 fake news detection algorithms 은 deceptive signals을 찾기 위해 mining news content, the surrounding exogenous context 에 초점을 맞췄다.
2. fake news를 전파하는 이용자의 endogenous preference 는 무시되었다.
3. Confirmation bias theory – 개인의 기존 신념(또는 성향)을 견고하게 하는 가짜 뉴스를 잘 퍼트린다는 심리학적 이론을 바탕으로 endogenous preference 에 주목.
4. 이용자의 historical, social engagement such as posts 는 users' preferences toward news 큰 정보를 제공하고 fake news detection 에 잠재적 요인이 된다.
5. Propose a new framework: UPFD that simultaneously captures various signals from user preferences by joint content and graph modeling.
6. Experimental results on real-world datasets demonstrate the effectiveness of the proposed framework.

In [1]:
```
import argparse
import os.path as osp
from tqdm.notebook import tqdm
```

In [2]:
```python
import torch
from torch.nn import Linear
import torch.nn.functional as F
from torch_geometric.datasets import UPFD
from torch_geometric.loader import DataLoader
from torch_geometric.transforms import ToUndirected
from torch_geometric.nn import GCNConv, SAGEConv, GATConv, global_max_pool, GraphConv, GINConv
```

```
C:\Anaconda3\lib\site-packages\torchvision\io\image.py:11: UserWarning: Failed to load image Python extension: Cou
ld not find module 'C:\Anaconda3\Lib\site-packages\torchvision\image.pyd' (or one of its dependencies). Try using
the full path with constructor syntax.
  warn(f"Failed to load image Python extension: {e}")
```

dataset - politifact, feature - spacy, model - GAT 예시로 사용

In [3]:
```python
# argparse - python 에 내장된 기능으로 argument 를 간단하게 활용할 수 있게 한다.
parser = argparse.ArgumentParser()
parser.add_argument('--dataset', type=str, default='politifact',
                    choices=['politifact', 'gossipcop'])
parser.add_argument('--feature', type=str, default='spacy',
                    choices=['profile', 'spacy', 'bert', 'content'])
parser.add_argument('--model', type=str, default='GAT',
                    choices=['GCN', 'GAT', 'SAGE', 'Graph_Conv'])
args = parser.parse_args('')

# UPFD dataset 다운로드
path = osp.join(osp.dirname(osp.realpath('./')), '..', 'data', 'UPFD')
train_dataset = UPFD(path, args.dataset, args.feature, 'train', ToUndirected())
val_dataset = UPFD(path, args.dataset, args.feature, 'val', ToUndirected())
test_dataset = UPFD(path, args.dataset, args.feature, 'test', ToUndirected())

train_loader = DataLoader(train_dataset, batch_size=128, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=128, shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=128, shuffle=False)
```

2. Dataset Description

1. UPFD dataset 은 트위터에서 발생한 fake & real 뉴스 선전 네트워크를 가지고 있다
2. 뉴스의 진위 여부는 Politifact 와 Gossipcop 에 의해 검증된 것이다
3. 뉴스의 retweet graphs 는 FakeNewsNet 에서 가져온 것이다

- retweet : 다른 사람의 트윗을 자신의 계정으로 그대로 다시 트윗하는 것

  즉 팔로잉하는 이용자의 트윗에 공감한 내용이 있을 때

  그것을 자신의 팔로워에게 전달할 때 사용한다

4. 데이터의 수집은 node features 생성을 위해 가짜 뉴스를 전파한

   사용자의 tweets(message) 의 기록을 추적한 것이다

Dataset statistics

Data	#Graphs	#Fake News	#Total Nodes	#Total Edges	#Avg. Nodes per Graph
Politifact	314	157	41,054	40,740	131
Gossipcop	5464	2732	314,262	308,798	58

5. 각 그래프는 hierarchical tree-structured graph 이다

   1) root node : the news

   2) leaf nodes : Twitter users who retweeted the root news

   3) User node 는 retweet 한 news node 에 edge 를 가진다

4) User 사이에 retweet 한 경우 edge 를 가진다

6. node feature types

1) bert - 768-dimensional

2) spacy - 300-dimensional

- pretrained BERT, spaCy word2vec 로 encoding 된 것이다

3) profile - 10-dimensional

4) content - 310-dimensional : 300-dim user comment word2vec(spaCy) + 10-dim profile

In [4]:
```
print(train_dataset)
print(val_dataset)
print(test_dataset)
```

```
UPFD(62, name=politifact, feature=spacy)
UPFD(31, name=politifact, feature=spacy)
UPFD(221, name=politifact, feature=spacy)
```

3. Model 구성

In [5]:
```
class Net(torch.nn.Module):
    def __init__(self, model, in_channels, hidden_channels, out_channels,
                 concat=False):
        super().__init__()
        self.concat = concat

        if model == 'GCN':
            self.conv1 = GCNConv(in_channels, hidden_channels)
        elif model == 'SAGE':
            self.conv1 = SAGEConv(in_channels, hidden_channels)
        elif model == 'GAT':
            self.conv1 = GATConv(in_channels, hidden_channels)
        elif model == 'Graph_Conv':
            self.conv1 = GraphConv(in_channels, hidden_channels)


        if self.concat:
            self.lin0 = Linear(in_channels, hidden_channels)
            self.lin1 = Linear(2 * hidden_channels, hidden_channels)

        # neural classifier
        self.lin2 = Linear(hidden_channels, out_channels)

    def forward(self, x, edge_index, batch):
        h = self.conv1(x, edge_index).relu()
        h = global_max_pool(h, batch)

        if self.concat:
            # Get the root node (tweet) features of each graph:
            root = (batch[1:] - batch[:-1]).nonzero(as_tuple=False).view(-1)
            root = torch.cat([root.new_zeros(1), root + 1], dim=0)
            news = x[root]

            # concatenation - exogenous information + endogenous information
            news = self.lin0(news).relu()
            h = self.lin1(torch.cat([news, h], dim=-1)).relu()

        h = self.lin2(h)
        return h.log_softmax(dim=-1)
```

concat = True 설정하여 endogenous information 을 input data 로 활용한다

In [6]:
```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = Net(args.model, train_dataset.num_features, 256,
            train_dataset.num_classes, concat=True).to(device)
                                    # concat=False - Only GNN ( exogenous information )
                                    # concat=True - adding endogenous information (user preference)
optimizer = torch.optim.Adam(model.parameters(), lr=0.001, weight_decay=0.01)
print(device)
```

```
cuda
```

train, validation, test loop 구성

In [7]:
```
def train():
    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 = F.nll_loss(out, data.y)
        loss.backward()
        optimizer.step()
        total_loss += float(loss) * data.num_graphs

    return total_loss / len(train_loader.dataset)
```

In [8]:
```
@torch.no_grad()
def test(loader):
    model.eval()

    total_correct = total_examples = 0
    for data in loader:
        data = data.to(device)
        pred = model(data.x, data.edge_index, data.batch).argmax(dim=-1)
        total_correct += int((pred == data.y).sum())
        total_examples += data.num_graphs

    return total_correct / total_examples
```

4. Evaluatation

In [9]:
```
for epoch in tqdm(range(0, 100)):
    loss = train()
    train_acc = test(train_loader)
    val_acc = test(val_loader)
    test_acc = test(test_loader)
    print(f'Epoch: {epoch:02d}, Loss: {loss:.4f}, Train: {train_acc:.4f}, '
          f'Val: {val_acc:.4f}, Test: {test_acc:.4f}')
```

```
Epoch: 59, Loss: 0.0752, Train: 1.0000, Val: 0.7742, Test: 0.7828
Epoch: 60, Loss: 0.0733, Train: 1.0000, Val: 0.7097, Test: 0.7828
Epoch: 61, Loss: 0.0718, Train: 1.0000, Val: 0.7742, Test: 0.7828
Epoch: 62, Loss: 0.0701, Train: 1.0000, Val: 0.7419, Test: 0.7828
Epoch: 63, Loss: 0.0687, Train: 1.0000, Val: 0.7097, Test: 0.7828
Epoch: 64, Loss: 0.0680, Train: 1.0000, Val: 0.7742, Test: 0.7828
Epoch: 65, Loss: 0.0671, Train: 1.0000, Val: 0.7419, Test: 0.7828
Epoch: 66, Loss: 0.0661, Train: 1.0000, Val: 0.7097, Test: 0.7828
Epoch: 67, Loss: 0.0656, Train: 1.0000, Val: 0.7742, Test: 0.7828
Epoch: 68, Loss: 0.0654, Train: 1.0000, Val: 0.7097, Test: 0.7828
Epoch: 69, Loss: 0.0648, Train: 1.0000, Val: 0.7742, Test: 0.7828
Epoch: 70, Loss: 0.0642, Train: 1.0000, Val: 0.7742, Test: 0.7828
Epoch: 71, Loss: 0.0640, Train: 1.0000, Val: 0.7097, Test: 0.7828
Epoch: 72, Loss: 0.0637, Train: 1.0000, Val: 0.7742, Test: 0.7873
Epoch: 73, Loss: 0.0633, Train: 1.0000, Val: 0.7097, Test: 0.7828
Epoch: 74, Loss: 0.0627, Train: 1.0000, Val: 0.7097, Test: 0.7828
Epoch: 75, Loss: 0.0623, Train: 1.0000, Val: 0.7419, Test: 0.7873
Epoch: 76, Loss: 0.0620, Train: 1.0000, Val: 0.7097, Test: 0.7783
Epoch: 77, Loss: 0.0616, Train: 1.0000, Val: 0.7419, Test: 0.7873
Epoch: 78, Loss: 0.0611, Train: 1.0000, Val: 0.7097, Test: 0.7783
```

5. GNN algorithm comparison

GNN UPFD 비교

1. 수행하는 Task는 Graph Classification(Fake / Real)
2. Fake news propagation과 real news propagation은 서로 다른 패턴이 있다는 통계적 결과를 바탕으로 실험
3. 추가적으로 Use preference information(user's historical posts)이 detection 성능 향상에 기여함을 강조함
4. UPFD_Politifact_spacy는 propagation-based 정보와 함께 user preference information(user의 최근 posts 내용을 spacy의 bag-of-words로 encoding한 것)을 detection에 활용함
5. UPFD_Politifact_non은 user preference information 없이 propagation-based 정보만 활용함
6. 단순한 lr 조정은 주어진 모델 안에서 loss의 발산 외 유의미한 차이를 가져오지 않음

Epochs : 100				Graph CLASSIFICATION			
				UPFD_Politifact_spacy		UPFD_Politifact_non	
Model	*L*	lr	c_hidden	Test Acc	Train Acc	Test Acc	Train Acc
MLP	1	0.001	32	79.19	98.39	69.23	96.77
			64	79.19	100.00	69.68	96.77
			128	78.73	100.00	71.04	96.77
			256	78.73	100.00	71.95	98.39
GCN	1	0.001	32	78.28	93.55	76.02	85.48
			64	80.09	96.77	81.00	91.94
			128	81.00	98.39	80.54	91.94
			256	83.71	98.39	81.45	95.16
GAT	1	0.001	32	76.02	96.77	68.78	88.71
			64	78.73	100.00	65.61	88.71
			128	79.19	100.00	77.83	95.16
			256	78.73	100.00	78.28	98.39
GraphConv	1	0.001	32	84.16	100.00	80.54	95.16
			64	83.71	100.00	79.19	93.55
			128	**84.62**	100.00	**81.90**	96.77
			256	83.71	100.00	80.54	100.00
SAGE	1	0.001	32	80.09	98.39	75.57	98.39
			64	80.54	100.00	78.28	98.39
			128	78.73	100.00	79.64	100.00
			256	79.64	100.00	77.38	100.00