Fake News Detection by using Graph Neural Network Model

22:544:647SP TPC: GRAPH METHODS & NETWORK ANALYSIS

Project By: Ankita Kukrety **Semester: Spring 2023**
Darshil Patel
Dhananjay Gaonkar
Siddhi Jadhav
Instructor: Spiros Papadimitriou **Number of credits: 3**

Introduction:

Our project aims to classify news propagation graphs on Twitter as either fake news or real news. This classification task is important because fake news and disinformation have harmful consequences on individuals and society, as they can spread rapidly and influence people's beliefs and actions based on false information. By identifying fake news graphs, our project can help to mitigate the spread of misinformation by alerting users to potentially misleading information and prompting them to fact-check before sharing or believing such information. This could help to prevent the harmful effects of fake news, such as the spread of conspiracy theories, the incitement of violence or social unrest, and the erosion of trust in institutions and the media. The UPFD dataset you are using contains information about the propagation of news on Twitter, and you will use this data to train a model that can accurately classify news propagation graphs as either real or fake news. This will require careful analysis of the graph structure, as well as consideration of other factors that may influence the spread of fake news on social media, such as user demographics, source credibility, and language use.

Overall, our project has the potential to make a significant contribution to the fight against fake news and disinformation, by providing a tool for early detection and intervention that can help to limit the harm caused by these phenomena.

Dataset:

The dataset we are using for fake news detection is based on Twitter data and contains both real and fake news labels. The news articles in the dataset have been fact-checked by PolitiFact and Gossip Cop, two websites that are dedicated to fact-checking news and information. The Twitter graph in the dataset was initially extracted by Fake Newsnet, which is available on GitHub. In this dataset, each news article serves as the root node of a graph. Twitter users who retweeted the news have an edge over the news, and each node in the graph has 310 dimensions. Of these 310 dimensions, 10 represent user features, and 300 represent the news embedding using the word-to-vector algorithm. The word-to-vector algorithm is a technique that converts words to numerical vectors, which can be used as inputs to machine learning models. By converting words to vectors, the algorithm can capture semantic relationships between words and learn representations that are useful for various natural languages processing tasks, such as text classification and sentiment analysis. In addition to edges between users and news articles, there are also edges between users who retweeted the same news article. This enables the model to capture the relationships and interactions between users who engage with the same news content.

Overall, this dataset provides a rich source of information for training and evaluating machine learning models for fake news detection on Twitter. The combination of news articles, user features, and interactions between users and news articles can help researchers develop more effective algorithms and techniques for identifying and combating the spread of fake news on social media.

Fake News Detection by using Graph Neural Network Model

22:544:647SP TPC: GRAPH METHODS & NETWORK ANALYSIS

Method Description:

Graph data can be analysed and used for predictions using machine learning techniques at different levels: node level, edge-level, and graph-level.

- At the node level, machine learning can be used to predict the behaviour or properties of individual nodes in a graph, such as whether a person will smoke or not smoke. This requires information about the characteristics and attributes of the node, as well as the relationships and interactions it has with other nodes in the graph.

- At the edge level, machine learning can be used to predict the likelihood of a relationship or interaction between two nodes in the graph, based on their attributes and previous interactions. For example, machine learning could be used to predict the next video that two users will like on Netflix, based on their viewing history and preferences.

- At the graph level, machine learning can be used to analyse the structure and properties of the entire graph, and make predictions or classifications based on that information. For example, in the case of fake news detection using a graph neural network, machine learning could be used to analyse the propagation of news articles and user interactions in a graph, and classify the graph as either real or fake news.

In our project, we are using machine learning to classify news propagation graphs on Twitter as either real or fake news. This involves analysing the structure and relationships of the nodes in the graph, as well as the interactions and characteristics of the users who retweeted the news articles. By using a graph neural network, you can leverage the power of graph analysis to make more accurate predictions and classifications than traditional machine learning techniques.

Installation and Importing:

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

import torch_geometric
```

PyTorch Geometric is a Python library for deep learning on graphs, built on top of PyTorch. To install PyTorch Geometric packages, you can use pip, which is a package installer for Python.

The packages can be installed by specifying the URL where the packages can be found. This URL typically points to a repository, such as GitHub or PyPI (Python Package Index), where the packages are hosted.

Fake News Detection by using Graph Neural Network Model

22:544:647SP TPC: GRAPH METHODS & NETWORK ANALYSIS

To install PyTorch Geometric packages using pip, you can open a terminal or command prompt and type in the following command:
pip install --verbose --no-cache-dir torch-scatter -f https://pytorch-geometric.com/whl/torch-1.9.0+cpu.html
This command installs the torch-scatter package from the PyTorch Geometric repository at the specified URL. The --verbose option displays detailed output during installation, while --no-cache-dir disables the pip cache to ensure that the latest version of the package is installed.

By installing PyTorch Geometric packages, you can use the library's powerful graph neural network algorithms and tools to analyse and manipulate graph data in your machine learning projects.

```
Torch vers:  2.0.0+cu118
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 107.6/107.6 kB 3.0 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
  Building wheel for torch-scatter (setup.py) ... done
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 209.2/209.2 kB 5.5 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
  Building wheel for torch-sparse (setup.py) ... done
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
  Building wheel for torch_geometric (pyproject.toml) ... done
```

torch-scatter, torch-sparse, and pytorch_geometric are dependencies required for PyTorch Geometric (PyG), a Python library for deep learning on graphs built on top of PyTorch.

torch-scatter provides a high-performance implementation of scatter operations on dense tensors. Scatter operations are used to accumulate values from one tensor into another tensor, based on the indices specified in a third tensor. These operations are commonly used in graph neural networks to aggregate node features across neighbourhoods.

torch-sparse provides a high-performance implementation of sparse tensor operations, which are useful for working with large graphs that have many sparse connections. Sparse tensor operations can significantly reduce the memory usage and computation time required for graph neural network training and inference.

pytorch_geometric provides a set of tools and utilities for loading, processing, and manipulating graph data. It includes data loaders for popular graph datasets, as well as various pre-processing and augmentation techniques. PyTorch Geometric also provides a variety of graph neural network models and building blocks, such as convolutional and recurrent layers, that can be easily combined and customized to suit specific graph learning tasks.
By installing these dependencies along with PyTorch Geometric, you can take advantage of the library's powerful graph neural network algorithms and tools to analyse and manipulate graph data in your machine-learning projects.

Building the wheel involves compiling the source code into a binary distribution format that can be installed using pip, the Python package installer. The torch-geometric library is installed from the GitHub repository by downloading the source code and building the wheel for installation. During the

build process, progress messages may be displayed indicating which parts of the library are being compiled and whether any errors or warnings have occurred. Once the build process is complete, the torch-geometric library can be installed using pip and its powerful graph neural network algorithms and tools can be used in PyTorch projects.

```
dataset = UPFD(root='.', name="gossipcop", feature='content')
train_data = UPFD(root=".", name="gossipcop", feature="content", split="train")
test_data = UPFD(root=".", name="gossipcop", feature="content", split="test")
print("Train Samples: ", len(train_data))
print("Test Samples: ", len(test_data))
```

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

The Gossip Cop dataset has been split into two subsets for training and testing purposes. The training subset contains 1092 data tests, which will be used to train the graph neural network model. The testing subset contains 3826 data tests, which will be used to evaluate the performance of the trained model. This process is known as data splitting and is a common practice in machine learning to prevent the model from memorizing the training data and to ensure that it can generalize well to unseen data.

```
train_data = UPFD(root=".", name="politifact", feature="content", split="train")
test_data = UPFD(root=".", name="politifact", feature="content", split="test")
print("Train Samples: ", len(train_data))
print("Test Samples: ", len(test_data))
```

```
Train Samples:  62
Test Samples:  221
```

The PolitiFact dataset has been split into two subsets for training and testing purposes. The training subset contains 62 data tests, which will be used to train the graph neural network model. The testing subset contains 221 data tests, which will be used to evaluate the performance of the trained model. This data-splitting process is essential for preventing the model from overfitting on the training data and ensuring that it can accurately classify fake news on unseen data.

Testing and Predictions:

```
from sklearn.metrics import accuracy_score, f1_score

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = GNN(train_data.num_features, 128, 1).to(device)
optimizer = torch.optim.Adam(model.parameters(), lr=0.001, weight_decay=0.001)
loss_fnc = torch.nn.BCELoss()
```

The test data is processed in mini-batches, which involves moving a small batch of data to the device (such as a GPU) for processing by the graph neural network model. The model predicts the labels for each sample in the mini-batch using the node features, edge indices, and batch indices. The predicted logits (scores) and labels, as well as the true labels, are then stored in a Pandas DataFrame. For example, the first ten samples in

each mini-batch are used to generate the predictions and store the results in the DataFrame. This process is repeated for each mini-batch until all of the test data has been processed and classified.

```
for epoch in range(20):
    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: 0.70 | TestLoss: 0.69 | TestAcc: 0.50 | TestF1: 0.00
Epoch: 01 |  TrainLoss: 0.70 | TestLoss: 0.69 | TestAcc: 0.50 | TestF1: 0.00
Epoch: 02 |  TrainLoss: 0.69 | TestLoss: 0.69 | TestAcc: 0.50 | TestF1: 0.02
Epoch: 03 |  TrainLoss: 0.69 | TestLoss: 0.68 | TestAcc: 0.62 | TestF1: 0.39
Epoch: 04 |  TrainLoss: 0.68 | TestLoss: 0.68 | TestAcc: 0.61 | TestF1: 0.37
Epoch: 05 |  TrainLoss: 0.68 | TestLoss: 0.68 | TestAcc: 0.83 | TestF1: 0.80
Epoch: 06 |  TrainLoss: 0.68 | TestLoss: 0.67 | TestAcc: 0.50 | TestF1: 0.67
Epoch: 07 |  TrainLoss: 0.67 | TestLoss: 0.66 | TestAcc: 0.55 | TestF1: 0.17
Epoch: 08 |  TrainLoss: 0.66 | TestLoss: 0.67 | TestAcc: 0.50 | TestF1: 0.01
Epoch: 09 |  TrainLoss: 0.65 | TestLoss: 0.66 | TestAcc: 0.50 | TestF1: 0.01
Epoch: 10 |  TrainLoss: 0.64 | TestLoss: 0.62 | TestAcc: 0.55 | TestF1: 0.19
Epoch: 11 |  TrainLoss: 0.60 | TestLoss: 0.59 | TestAcc: 0.55 | TestF1: 0.69
Epoch: 12 |  TrainLoss: 0.54 | TestLoss: 0.51 | TestAcc: 0.65 | TestF1: 0.47
Epoch: 13 |  TrainLoss: 0.41 | TestLoss: 0.28 | TestAcc: 0.97 | TestF1: 0.97
Epoch: 14 |  TrainLoss: 0.20 | TestLoss: 0.15 | TestAcc: 0.96 | TestF1: 0.96
Epoch: 15 |  TrainLoss: 0.14 | TestLoss: 0.12 | TestAcc: 0.97 | TestF1: 0.97
Epoch: 16 |  TrainLoss: 0.15 | TestLoss: 0.12 | TestAcc: 0.96 | TestF1: 0.96
Epoch: 17 |  TrainLoss: 0.12 | TestLoss: 0.10 | TestAcc: 0.97 | TestF1: 0.97
Epoch: 18 |  TrainLoss: 0.11 | TestLoss: 0.11 | TestAcc: 0.98 | TestF1: 0.98
Epoch: 19 |  TrainLoss: 0.12 | TestLoss: 0.10 | TestAcc: 0.97 | TestF1: 0.97
```

	pred_logit	pred	true
0	0.990395	1.0	1
1	0.013238	0.0	0
2	0.034529	0.0	0
3	0.993312	1.0	1
4	0.942137	1.0	1
5	0.006301	0.0	0
6	0.005818	0.0	0
7	0.989767	1.0	1
8	0.971058	1.0	1
9	0.840354	1.0	1

The output log provides information on the model's prediction for each sample in the test data. In the first row of the log, the model predicted a probability of 0.99 for the sample, indicating a high degree of confidence in its classification. The predicted label for this sample is 1, which means the model classified it as a fake news sample. The true label is also 1, indicating that this sample is actually a fake news sample. This suggests that the model was able to accurately classify this sample.

Result and Discussions:

Higher values for TestAcc and TestF1 indicate better performance of the model in classifying the test data. The model's accuracy of 97% suggests that it has performed very well in detecting fake news. Graph Neural Networks are found to be more effective in detecting fake news compared to traditional Natural Language Processing (NLP) models. In particular, Graph Convolutional Networks (GCN) and Attention-based Graph Neural Networks are very effective and perform comparably well. Overall, this suggests that Graph Neural Networks can be a promising approach for fake news detection.

Fake News Detection by using Graph Neural Network Model

22:544:647SP TPC: GRAPH METHODS & NETWORK ANALYSIS

Summary and Conclusion:

Graph Neural Networks (GNNs) have been found to be effective in detecting fake news by leveraging the graph structure of the data. The graph structure of fake news data can be used to capture complex interactions between various entities and features in a more accurate and robust way compared to traditional machine learning models. By identifying patterns and relationships in the graph structure of fake news data, GNNs can effectively detect fake news.

In conclusion, the model uses PyG to load the UPFD dataset and converts it to NetworkX graphs. It then creates data loaders for training and testing, defines and trains a three-layer Graph Neural Network model on the dataset, and evaluates the performance of the model on the test set. The model is able to effectively capture the complex interactions between various entities and features in fake news data, which results in more accurate and robust detection of fake news compared to traditional machine learning models. The evaluation shows that the model performs well with an accuracy of 97%, which indicates that the model has detected fake news efficiently.