Impact of graph and node-level features on Fake News Detection

Victor Goubet - 2039343
UNIPD
Computer engineering

Nicholas Tagliapietra - 1242481
UNIPD
Mathematical Engineering

Asma Bakhtiariazad - 2050788
UNIPD
ICT

January 10, 2022

Github: FND

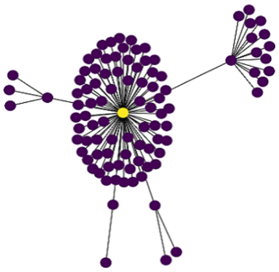
1 Abstract

Fake news are the new plague of the 21st century. With the advent of social networks and the easy and quick access to information, this disease has become more and more common. Through retweets, shares and likes, a piece of fabricated information can in a few moments gain real credibility thanks to the common ”validation”. Similar to the tragedy of the common, each user is selfish and does not take the time to verify the sources, preferring to believe in this information that is often incredible, revolutionary and built to make the buzz. However, for several years now, various methods have been used on social networks to address these problems by detecting and removing problematic messages from the platform as quickly as possible. Our objective is to analyse the state of the art of these methods, to implement a graph-based solution and to see if adding node-level features helps to increase the prediction score. We will take the example of Twitter and model a graph for each tweet posted. Finally, we will use the a version of the FakeNewsNet dataset [5] which is one of the reference dataset for this type of tasks.

2 Dataset

We will use a different version of the FakeNewsNet dataset called UPFD[7]. The advantages of this Framework is that it is available directly in Pytorch Geometric, that it doesn't need Twitter APIs (which in our case were very limiting), and also that various node-level features and embeddings have been already computed thanks to different techniques explained in [7].

The Dataset is composed of several thousands graphs in the form of a tree where the root node is the source news and all the linked nodes correspond to the users that tweeted the news. The edges represent the sharing history between users (not between each tweet).



Here we recognize the news in yellow and each user who retweeted in purple. The goal is to solve a graph classification task where each graph must be classified into fake-news or not-fake-news. The dataset is divided in Gossip News (Gossipcop) and Political News (Politifact). When loading the dataset we can choose freely which part to use (Gossipcop) and also what node-level features to adopt (user profile info, embedded history of the user's tweets). The node-features had been extracted and precomputed in [7] thanks to the Spacy Word2vec encoder or BERT. Finally, the dataset contains 5464 graphs corresponding to real news and 2732 fake news, each graph being on average composed of 58 nodes (A).

3 Intended experiments

The goal is simple: study how the graph-level and node-level features contribute to the final result. We will extract new features and validate/reject all of them by testing them on different models.

As seen before, the UPFD Framework provides various node-level features:

- **Profile feature**: 10 node features of user profile attributes.
- **Spacy features:** 300 node features of user historical tweets [1]
- **BERT feature**: 768 node features of user historical tweets [6]

Node-level features

The node-level features that will be tested are: [**Degree, Closeness centrality, Betweenness centrality**]. Once these new features are computed, we can merge them to the existing features vector. Different combinations and fusion of features will be made in order to maximize the results (e.g. by comparing the concatenation or the multiplication of some of them).

Graph-level features

For each graph the new calculated features are: [**Average degree, Standard deviation degree, Degree of the root node, Closeness centrality of the root node, Maximal closeness centrality, Maximal betweenness centrality**]. Features fusion could be done just before the linear layer of a GNN (e.g. after a mean-max pooling) but, as before, different strategies will be tested.

Finally, various models will be built (GCN[4], GraphSAGE [3], GAT [8], et similia.) each one being different but inspired on the UPFD architecture (B). Then, these models will be trained with and without different features, and then evaluated with respect to different metrics. In such a way the most relevant features and the best models will be identified. Everything will be developed with Pytorch Geometric [2] and NetworkX.

4 Features computation and dataset creation

In order to calculate the different features, the *networkx* package has been used. Each graph is relatively small ($\forall i \in \llbracket 1 : n \rrbracket : 2 \leq |E_i| \simeq |N_i| \leq 200$) with n the number of graphs, hence it is possible to calculate the exact values of the features without using approximated algorithms.
In a second step, the default UPFD class of the *pytorch_geometric* has been modified in order to add the new features. When processing the data, the new node features are added to the existing ones (the profile feature vector containing the default UPFD features of length 10 described above) contained in the feature matrix X, while the graph features are added to the label matrix Y.

So, for each graph g_i, we have:

$$X_i = \begin{bmatrix} p_{1,1} & p_{1,2} & \cdots & p_{1,10} & x_{1,11} & \cdots & x_{1,10+a} \\ p_{2,1} & p_{2,2} & \cdots & p_{2,10} & x_{2,11} & \cdots & x_{2,10+a} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ p_{|N_i|,1} & p_{|N_i|,2} & \cdots & p_{|N_i|,10} & x_{|N_i|,11} & \cdots & x_{|N_i|,10+a} \end{bmatrix}$$

$$with \quad \begin{cases} p_{j,k} : k^{th} \text{ profile-feature of node j for graph i} \\ x_{j,k} : k^{th} \text{ new node-feature of node j for graph i} \\ a : \text{number of new node-feature} \end{cases}$$

and

$$Y_i = \begin{bmatrix} l & y_1 & \cdots & y_b \end{bmatrix} \quad with \quad \begin{cases} l : \text{label for graph } i \\ y_k : k^{th} \text{ new graph-feature of graph i} \\ b : \text{number of new graph-feature} \end{cases}$$

Both the X_i vector and the y_i values are normalized by using a max-normalization which rescale every single feature in the interval $[-1; 1]$. Once the custom class was created we could instantiate three different datasets:

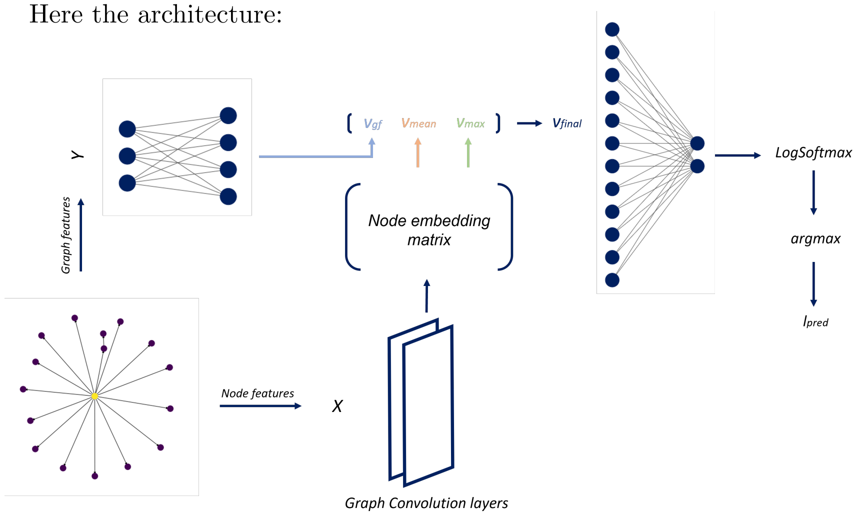
	Node feature			Graph feature					
	dg	cc	bc	avg(dg)	std(dg)	root_dg	root_cc	max(cc)	max(bc)
Original									
Dataset1	X	X		X	X				
Dataset2	X	X	X	X	X	X	X	X	X

5 Modeling and optimisation

Concerning the models, three models have been created in order to compare the results and the impact of the new feature on different architectures. The first one is a classic GNN composed of two graph convolution layers, the second one is an implementation of GraphSAGE[3] and the last one is GAT[8]. For GraphSAGE two SAGE layers have been used, associated with *relu* activation and dropout with $p = 0.5$. For GAT, two graph attention network layers are used in concatenation.

For the implementation of graph-level features it has been used a linear layer, taking a vector of graph-level features and transforming it into a vector v_{gf} of length n_e which is the latent dimension. After passing the x node-features matrix through the convolutional layers, a mean and max pooling have been applied in order to get respectively the vectors v_{mean} and v_{max}. Then, v_{mean}, v_{max} and v_e (v_e do not exists on the original dataset) are concatenated into a vector v_f, which is passed through a linear layer with 2 nodes (fake/real) and then a LogSoftmax activation function. Finally, the predicted class has been chosen by taking the *argmax* of the final vector.

Here the architecture:



Remark: The vectors v_{gf}, v_{mean} and v_{max} are vectors in $\mathbb{R}^{n_e}$, thus $v_f \in \mathbb{R}^{3n_e}$ if the new graph-level features are used, or $v_f \in \mathbb{R}^{2n_e}$ if not (when using the original dataset). Moreover, different approaches for vector aggregation (addition, multiplication, with and without a linear layer) have been tried and the most relevant was the simple concatenation.

Training and hyperparameter optimization The training made use of the adaptive moment estimation algorithm (Adam) for all the models, where the learning rate, weight decay and batch size had to be tuned via an hyperparameter optimization procedure. Regarding the loss, the negative log likelihood loss [9] has been used along with the LogSoftmax activation. This combination is known to add bigger penalty on large errors, to have a cheaper training cost and to be more stable numerically than using classical softmax activation with likelihood loss.

The UPFD Framework already provides the dataset splitted in training, validation and test set, and the best hyperparameters have been chosen in a cross-validation setup: all the architectures have been trained by using the training set, selected by evaluating the performance on the validation set and tested on the test set. This procedure has been implemented thanks to the *Optuna* library that provide various hyperparameters search algorithms, a pruning mechanism for stopping unpromising architectures during training and also an early stopping mechanism that enabled the search to be executed in a reasonable time. Due to computational resources, for each model and dataset combination at maximum 40 different trials (Suggested by the Optimizer) have been done, which however has not been limiting.

The hyperparameters that have been tuned are the dimension of the embedding space (same for the two Layers), the learning rate, the weight decay and the batch size. At the end of the training, each of the 3 different datasets have been optimized and tested on the 3 dataset variations. In total 3 architectures × 3 datasets = 9 trained models are provided.

6 Results

Three different combinations of additional features and three different classifiers have been tried. The performance of each model has been assessed with different metrics: accuracy on the test set, true positive rate (*threshold* = 0.5), and A.U.C. (Area Under Curve). Receiver Operating Characteristic (ROC) plot and TPR/FPR plot vs. threshold have also been provided (D/E/F/G).

It can be seen on the results that the additional features improved the graph classification task. Moreover, by looking at the best models obtained on each dataset, it seems that the best one proved to be the first custom dataset (*dataset1* containing the degree and closeness centrality for each node and the average degree and root degrees for each graph.

	Origin			**Dataset 1**			**Dataset 2**		
	GNN	SAGE	GAT	GNN	SAGE	GAT	GNN	SAGE	GAT
Test accuracy	0.909	0.929	0.901	0.922	0.951	0.932	0.919	0.925	0.934
AUC	0.964	0.983	0.971	0.972	0.985	0.976	0.970	0.978	0.980
TPR 0.5	0.891	0.888	0.886	0.907	0.746	0.898	0.908	0.897	0.879
mean	0.921	0.933	0.919	0.934	0.894	0.935	0.932	0.933	0.931

It's important to note that optimizations and improvements like hyperparameter optimization, the use of additional graph level features or features normalization proved to be crucial for the correct functioning of the models, and without them a heavy loss of performance could be observed.

The data show that the new combinations of features allow to improve the effectiveness of fake new detection. However, this improvement is small and should be confirmed on more different datasets to be sure that it is significant. Moreover, we can wonder why do we have this degradation on the second dataset. This phenomenon can be explained by looking at the structure of the single networks, which are in the majority of cases composed of a root node connected to tens of leafs, so the networks did not present a very complex structure that could be described by the new features and even by more advanced ones like graphlet motifs. Indeed, the new features are all strongly correlated between them (C), so they add no useful information to the GNNs and sometimes can be even deleterious. It is what it happens with the *dataset2*: we added new features correlated to those in *dataset1* and therefore we just degraded the performance by adding new parameters in the model for free.

Another problem that limited the performances is the limited and fragmented amount of data available: since social networks are continuously involved in banning users that spread misinformation and deleting posts and tweets that contain fake news, many of the patterns contained in the dataset are incomplete graphs with missing nodes (banned tweets) that do not represent the full sharing history of the fake news. Having a dataset containing the information of every post/tweet and user involved in the spread of a fake news (including banned content) could help the performance of the classifiers. Finally, and probably the most important remark here is that some dynamic features like the timeline of each tweet could add a temporal dimension to the problem and help to describe the dynamics of how news are spreading over the internet i.e. its virality.

7 Participations

The development of the code initially started by using individual notebooks, which then have been merged and organized into a single codebase contained in the GitHub Repository presented in this report. The individual contributions are listed as follows:

Victor Goubet:
- Creation of the extended datasets
- Elaboration of the main architecture
- Creation of the Basic GNN model
- Creation of the SAGE model
- Development of the final notebook with plots
- Writing of the reports
- Debugging

Nicholas Tagliapietra:
- Creation of the extended datasets
- First version of Basic GNN
- Creation of the GAT model
- Development of Hyperparameter Optimization with Optuna
- Writing of the reports
- Debugging

Asma Bakhtiariazad: //

References

[1] Tomas Mikolov Kai Chen Greg Corrado and Jeffrey Dean. “Efficient estimation of word representations in vector space”. In: (2013).

[2] Matthias Fey and Jan E. Lenssen. “Fast Graph Representation Learning with PyTorch Geometric”. In: (2019).

[3] William L. Hamilton, Rex Ying, and Jure Leskovec. “Inductive Representation Learning on Large Graphs”. In: (2017).

[4] Thomas N. Kipf and Max Welling. “Semi-Supervised Classification with Graph Convolutional Networks”. In: (2016). arXiv: 1609.02907.

[5] Kai S. Deepak M. Suhang W. Dongwon L. “FakeNewsNet: A Data Repository with News Content, Social Context and Spatialtemporal Information for Studying Fake News on Social Media”. In: (2018).

[6] Jacob Devlin Ming-Wei Chang Kenton Lee and Kristina Toutanova. “Bert: Pre-training of deep bidirectional transformers for language understanding”. In: (2018).

[7] Yingtong D. Kai S. Congying X. Philip S. Lichao S. “User Preference-aware Fake News Detection”. In: (2021).

[8] Petar Veličković et al. “Graph Attention Networks”. In: (2018).

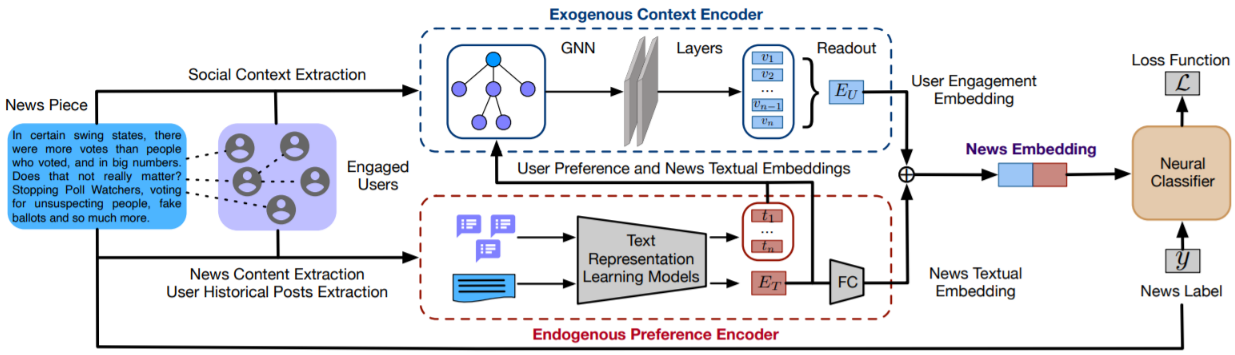
[9] Donglai Zhu et al. *Negative Log Likelihood Ratio Loss for Deep Neural Network Classification.* 2018. arXiv: 1804.10690 [cs.LG].

A Dataset statistics

Dataset	Graphs	Total	Total	Avg.
	(Fake)	Nodes	Edges	Nodes/Graph
Politifact (POL)	314 (157)	41.054	40.740	131
Gossipcop (GOS)	5464 (2732)	314.262	308.798	58

Table 1: Statistics of the two datasets

B Architecture of Fake News Detection Network in UPFD



Exogenous Context Encoder
GNN
Layers
Readout
v_1
v_2
...
v_{n-1}
v_n
E_U
Social Context Extraction
News Piece
In certain swing states, there were more votes than people who voted, and in big numbers. Does that not really matter? Stopping Poll Watchers, voting for unsuspecting people, fake ballots and so much more.
Engaged Users
User Engagement Embedding
Loss Function
$\mathcal{L}$
News Embedding
Neural Classifier
User Preference and News Textual Embeddings
Text Representation Learning Models
t_1
...
t_n
E_T
FC
News Textual Embedding
y
News Label
News Content Extraction
User Historical Posts Extraction
Endogenous Preference Encoder

C Features correlation

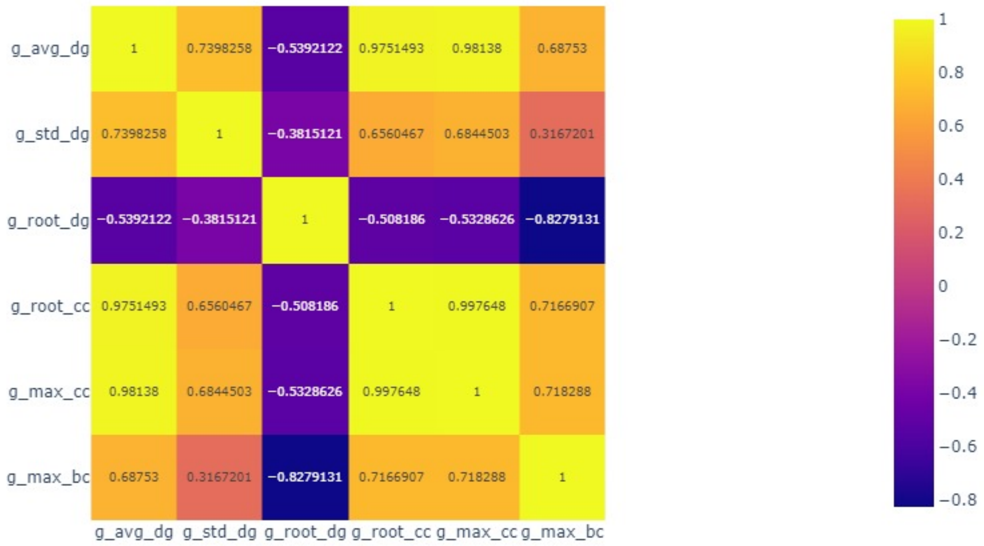


Figure 1: Correlation feature of graphs features

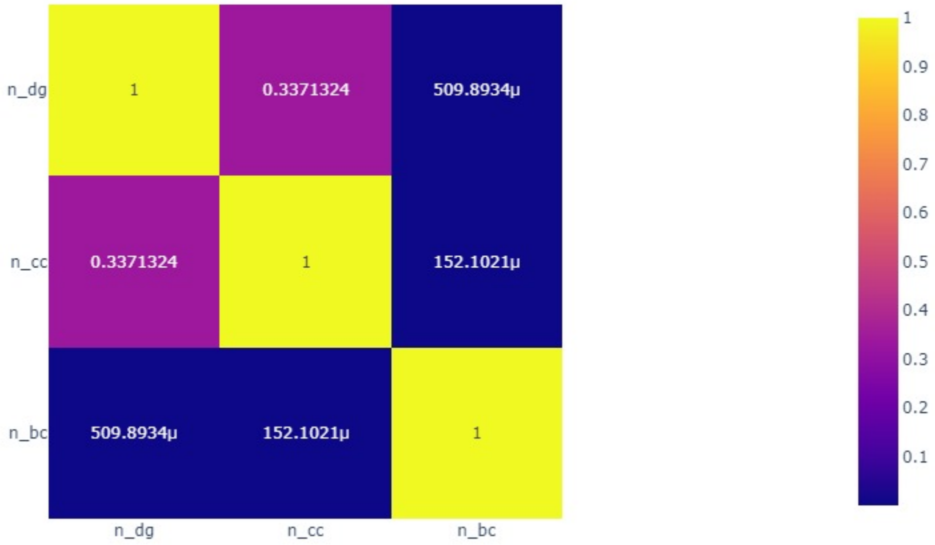
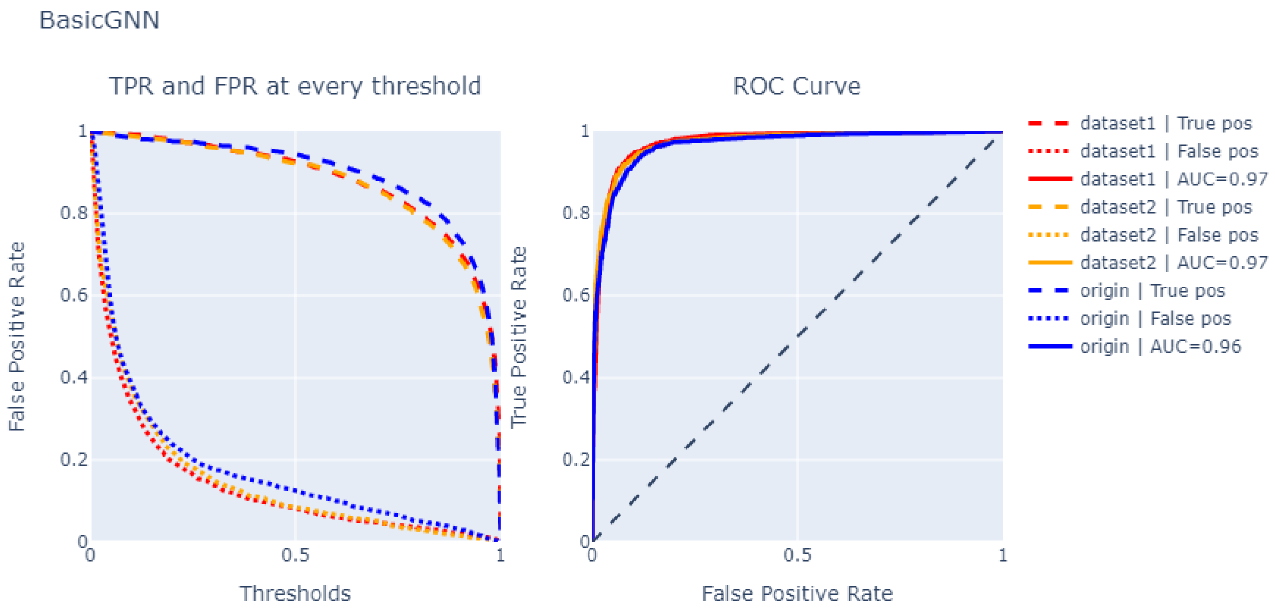
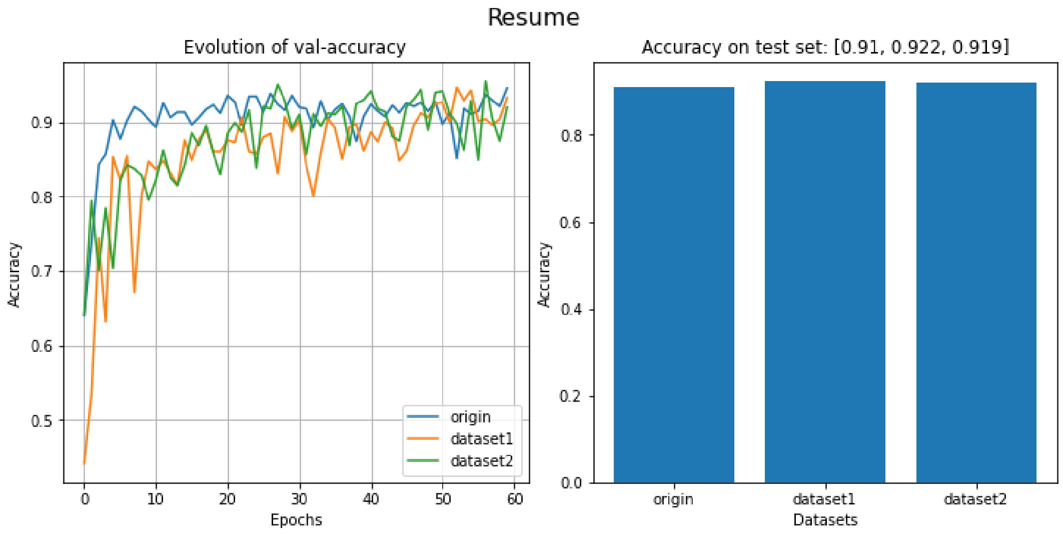


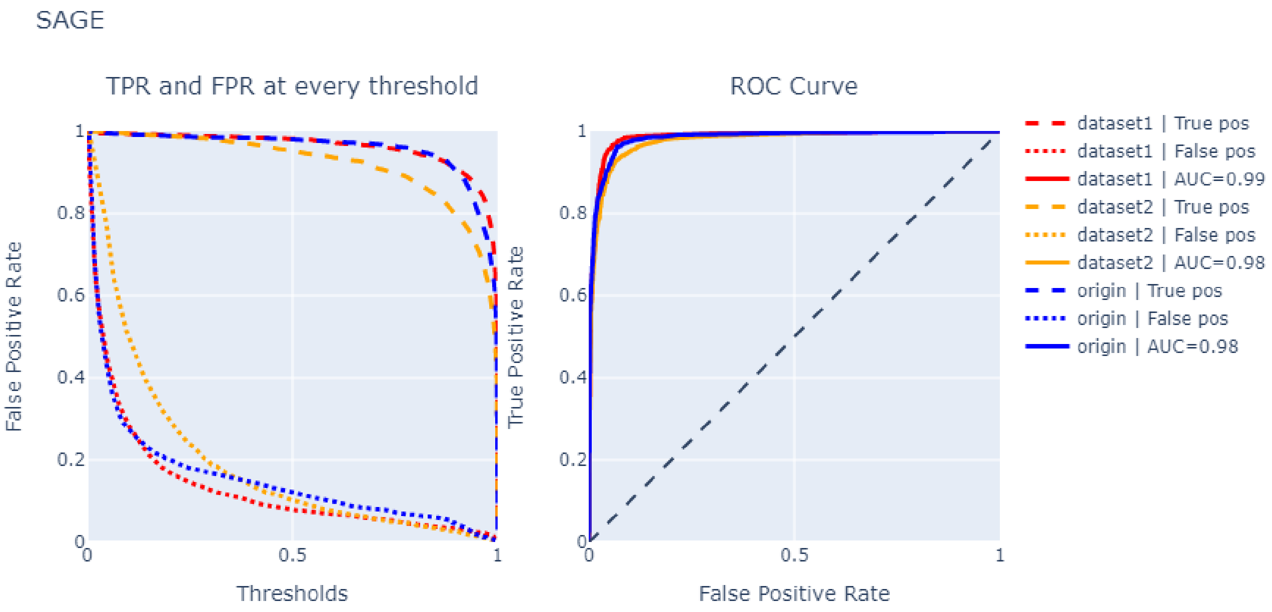
Figure 2: Correlation feature of nodes features

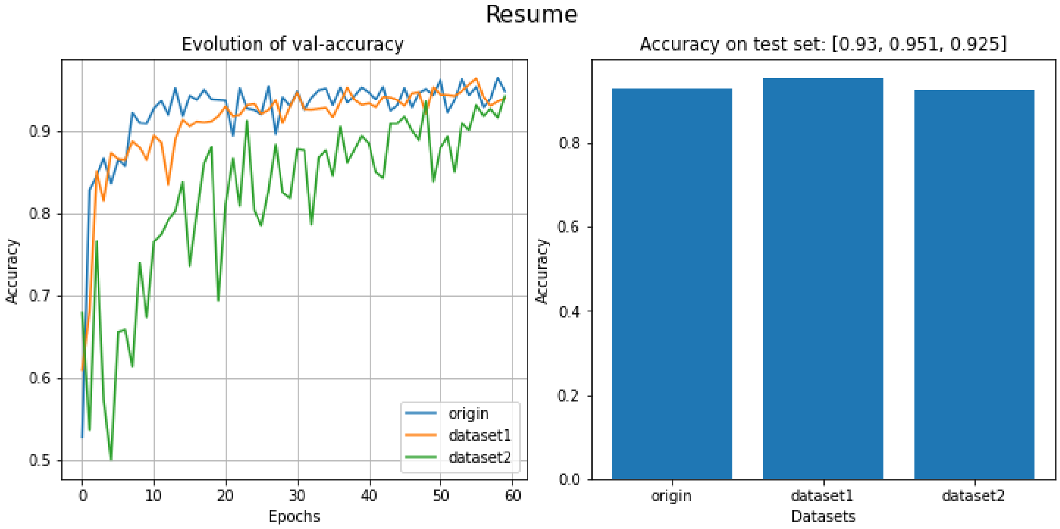
D GNN results



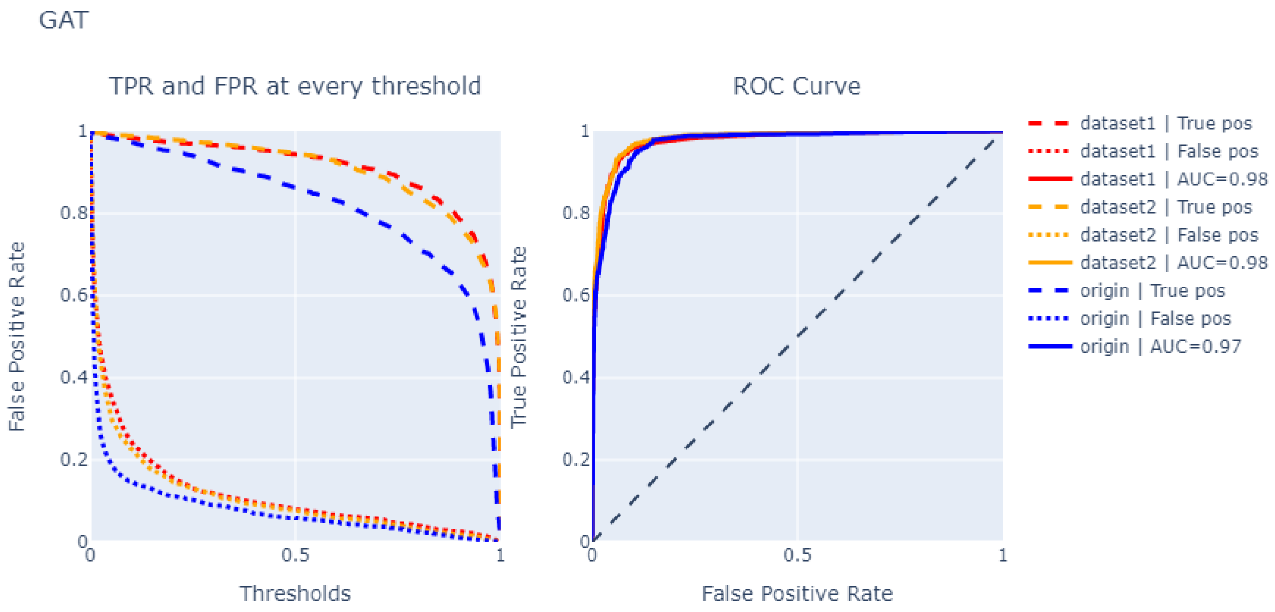


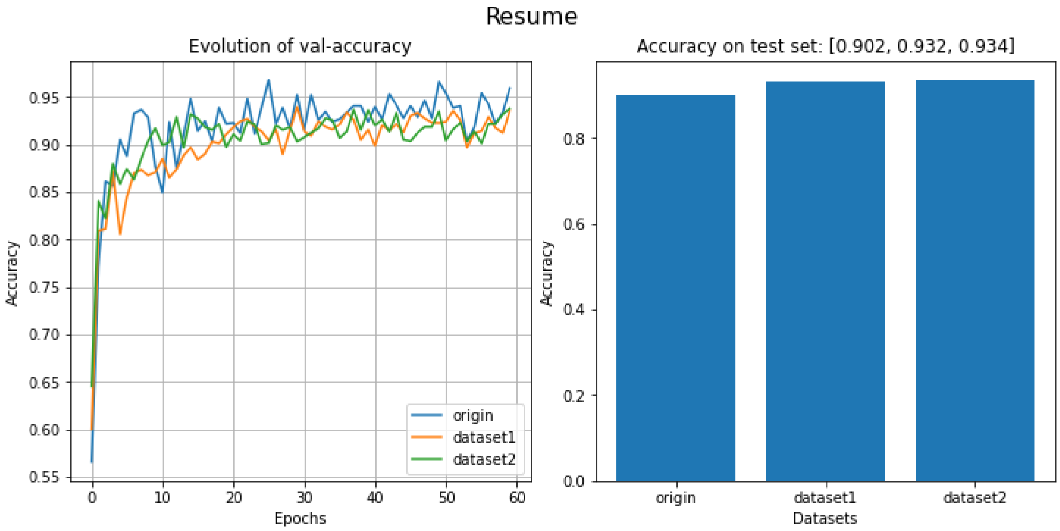
E SAGE Results





F GAT results





G Global results

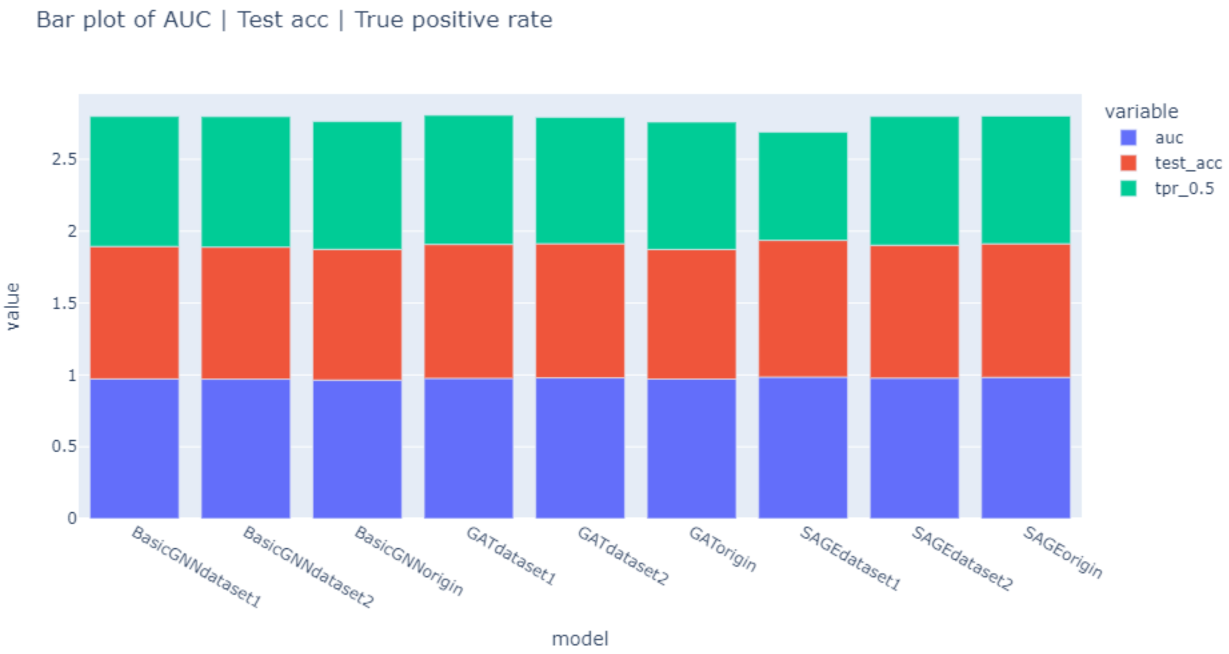


Figure 3: Bar plot of AUC, test accuraccy and true positive rate ($threshold = 0.5$) for each model-dataset.

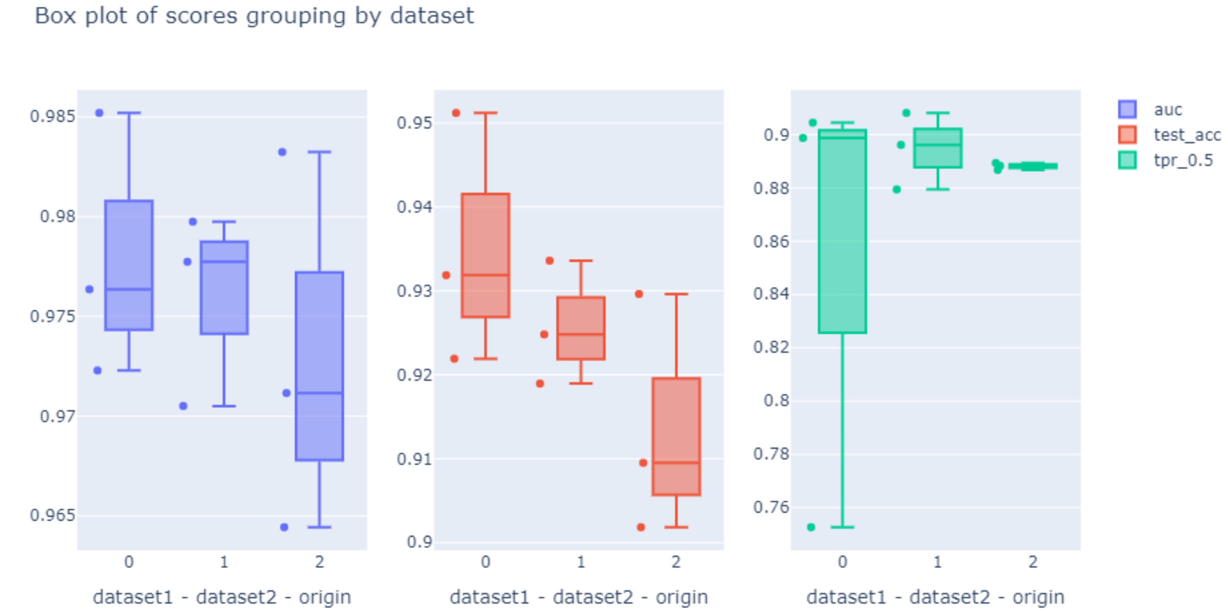


Figure 4: Box plot of the AUC, test accuraccy and true positive rate ($threshold = 0.5$), grouped by dataset.