Social Networks and Online Markets
Homework 1

Cristiana Di Tullio

May 2024

Contents

Problem 1

Consider a generalization of the Barabási–Albert preferential attachment model studied in the course. When a new node arrives at time t again it comes with ℓ edges. However, this time each edge selects a node v with probability proportional to the degree d_v plus a constant c, that is, the probability equals where the probability of a node v being selected as the endpoint of a new edge is:

$$\frac{d_v + c}{(t-1)(2\ell + c)}, \qquad \text{where} \quad c \geq -\ell$$

(so for c = 0 this is the Barabási–Albert model). Show that the degree distribution that we obtain as $t \to \infty$ is approximately a power law with exponent $3 + c/\ell$.

Recall that a for a random variable to follow a *power law distribution* with exponent $\gamma > 0$, the probability that it obtains a value x must be proportional to $x^{-\gamma}$. Power law distributions belong to the class of heavy tail distributions and the "heaviness" of the tail is specified by the parameter γ.
First of all, let's define:

- $n_k(t)$: number of nodes with degree k at time t;
- $p_k(t) = \frac{n_k(t)}{t}$: normalized frequency of nodes with degree k at time t;
- $\frac{k+c}{t(2+c/\ell)}$: probability that a node increases its degree from k to $k+1$ at time $t+1$. It is obtained multiplying the given probability of a node to be selected as endpoint for a new edge by the number ℓ of new edges coming at each time instant. In fact,

$$\ell \cdot \frac{d_v + c}{t(2\ell + c)} = \cancel{\ell} \cdot \frac{d_v + c}{\cancel{\ell} t(2 + c/\ell)} = \frac{d_v + c}{t(2 + c/\ell)} = \frac{k + c}{t(2 + c/\ell)}$$

For different values of k, we write the following expressions for $n_k(t+1)$:

$$\begin{cases} n_k(t+1) = 0 & \text{for } k > \ell \\ n_k(t+1) = n_k(t) + n_{k-1}(t)\frac{k-1+c}{t(2+c/\ell)} - n_k(t)\frac{k+c}{t(2+c/\ell)} & \text{for } k > \ell \\ n_l(t+1) = n_\ell(t) + 1 - n_\ell(t)\frac{\ell+c}{t(2+c/\ell)} & \text{for } k = \ell \end{cases}$$

The number of nodes of degree k at time $t+1$ is expressed as the number of nodes already having degree k at time t, plus the expected number of nodes with degree $k-1$ at time t that will become with degree k at time $t+1$, minus the expected number of nodes with degree k that will increase their degree by 1 at time $t+1$. In the first case, our quantity of interest is 0 as it can never happen that the degree of a node is lower than ℓ (every new node comes in the graph with exactly ℓ edges, so this represents the minimum node degree).
We can now rewrite the above expressions in terms pf $p_k(t)$:

$$\begin{cases} (t+1)p_k(t+1) = 0 & \text{for } k > \ell \\ (t+1)p_k(t+1) = tp_k(t) + p_{k-1}(t)\frac{k-1+c}{2+c/\ell} - p_k(t)\frac{k+c}{2+c/\ell} & \text{for } k > \ell \\ (t+1)p_\ell(t+1) = tp_\ell(t) + 1 - p_\ell(t)\frac{\ell+c}{2+c/\ell} & \text{for } k = \ell \end{cases}$$

We want to study the long-term behaviour of $p_k(t)$ and see if it follows a power law distribution. In order to do so, we assume that when $t \to \infty$, the quantity $p_k(t)$ will converge to a certain value p_k independent of t.
Therefore, we can consider:

$$p_k(t+1) = p_k(t) = p_k$$

Rewriting again:

$$\begin{cases} p_k = 0 & \text{for } k > \ell \\ p_k = p_{k-1}\frac{k-1+c}{k+2+c+c/\ell} & \text{for } k > \ell \\ p_\ell = \frac{2+c/\ell}{\ell+2+c+c/\ell} & \text{for } k = \ell \end{cases}$$

We will now focus on the middle expression, as it represents the general case $k \geq \ell$, to show that the degree distribution follows a power law with the desired exponent. Proceeding to the recursive equation as seen in class, although, we get to an equation that is not straightforward to simplify. Therefore, we change approach and proceed setting up a continuum equation.

$$p_k = p_{k-1}\frac{k-1+c}{k+2+\frac{c(1+\ell)}{\ell}}$$

$$p_k[k+2+\tfrac{c(1+\ell)}{\ell}] = p_{k-1}(k-1+c)$$

$$kp_k + 2p_k + \tfrac{c(1+\ell)}{\ell}p_k = kp_{k-1} - p_{k-1} + cp_{k-1}$$

$$kp_k - kp_{k-1} = -p_{k-1} + cp_{k-1} - 2p_k - \tfrac{c(1+\ell)}{\ell}p_k$$

$$k\tfrac{\partial p_k}{\partial k} = p_k(c-1-2-\tfrac{c(1+\ell)}{\ell})$$

$$k\tfrac{\partial p_k}{\partial k} = p_k(c-3-\tfrac{c(1+\ell)}{\ell})$$

$$\tfrac{\partial p_k}{p_k} = (c-3-\tfrac{c(1+\ell)}{\ell})\tfrac{\partial k}{k}$$

Integrating:

$$\ln(p_k) = \ln(k)^{(c-3-\frac{c(1+\ell)}{\ell})} + c_1$$

Passing to the exponential:

$$p_k = c_2 \cdot k^{(c-3-\frac{c(1+\ell)}{\ell})}$$

Note: c_1 and c_2 are just constants, we can ignore them for our purposes.
We need to focus on the exponent of k:

$$c - 3 - \frac{c(1+\ell)}{\ell} = \frac{\ell c - 3\ell - c - \ell c}{\ell} = \frac{-3\ell - c}{\ell} = -(3 + \frac{c}{\ell})$$

Thus, the degree distribution is indeed a power law distribution where the probability of obtaining a degree k is proportional to $k^{-\gamma}$ with $\gamma = (3 + \frac{c}{\ell})$.

Problem 2

See the notebook `Problem2.ipynb` for the implementation.

Problem 3

An interesting phenomenon in social networks is that a random person's expected degree is smaller than the degree of her peers: "Your friends are more popular than you are!" Given an undirected graph $G = (V, E)$, select a random node and let X be the random variable that equals to its degree and Y to be the random variable that equals the average degree of the node's neighbors.

1. Prove that $\mathbb{E}[X] \leq \mathbb{E}[Y]$.
2. When do we have that $\mathbb{E}[X] = \mathbb{E}[Y]$?

Hint. Prove that for any $a, b \neq$ we have that $\frac{a}{b} + \frac{b}{a} \geq 2$.

Given a node v chosen uniformly at random and its neighborhood $N(v)$, we define the average number of its neighbours as:

$$\mu = \mathbb{E}[X] = \sum_{v \in V} \frac{d_v}{|V|}$$

The average number of neighbours of the neighbours of v can now be modeled in two steps. First, we choose a random edge in the graph and a random endpoint of this edge. the probability that the node v is chosen is:

$$\frac{d_v}{|E|} \frac{1}{2}$$

Then we proceed to calculate the expected number of friends of a random friend of v:

$$\mathbb{E}[Y] = \sum_{v \in V} (\frac{d_v}{|E|} \frac{1}{2}) d_v = \sum_{v \in V} \frac{d_v^2}{2|E|}$$

Now, the variance of the degree d_v can be defined as the squared deviations from the mean degree:

$$\sigma^2 = \frac{1}{|V|} \sum_{v \in V} (d_v - \mu)^2$$

Expanding the squared term and doing some computation, we get:

$$\sigma^2 = \frac{1}{|V|} \sum_{v \in V} d_v^2 - \mu^2$$

Therefore, we can rewrite:

$$\frac{1}{|V|} \sum_{v \in V} d_v^2 = \mu^2 + \sigma^2$$

And substituting in the expression of $\mathbb{E}[Y]$, we get:

$$\mathbb{E}[Y] = \sum_{v \in V} \frac{d_v^2}{2|E|} = \frac{|V|}{2|E|} (\mu^2 + \sigma^2) = \mu + \frac{\sigma^2}{\mu}$$

Now we can compare $\mathbb{E}[X]$ and $\mathbb{E}[Y]$ and their relation:

$$\mathbb{E}[X] \leq \mathbb{E}[Y]$$
$$\mu \leq \mu + \frac{\sigma^2}{\mu}$$

Since $\frac{\sigma^2}{\mu}$ is always non-negative, the inequality is satisfied and obtained that the average degree of a neighbour of v is greater than the average degree of v itself.
We have that $\mathbb{E}[X] = \mathbb{E}[Y]$ when:

$$\mu = \mu + \frac{\sigma^2}{\mu}$$

That implies $\sigma^2 = 0$. Intuitively, the degree variance is 0 when all degrees are the same, i.e. when the graph is *regular*. Each node in this kind of graphs has exactly the same number of neighbours, so the degree of a node and the average degree of its neighbours will coincide.
An alternative approach is to express $\mathbb{E}[Y]$ as follows:

$$\mathbb{E}[Y] = \sum_{v \in V} \frac{1}{|V|} \sum_{u \in N(V)} \frac{d_u}{|N(v)|} = \sum_{v \in V} \frac{1}{|V|} \sum_{u \in N(V)} \frac{d_u}{d_v}$$

If the graph is *d-regular*, the expected value of the degree of a random node is trivially $\mathbb{E}[X] = d$. Since $d_v = d_u = d$, we can check that:

$$\mathbb{E}[Y] = \sum_{v \in V} \frac{1}{|V|} \sum_{u \in N(V)} \frac{d}{d} = \sum_{v \in V} \frac{1}{|V|} \sum_{u \in N(V)} 1 = \sum_{v \in V} \frac{1}{|V|} d =$$

$$= \frac{1}{|V|} \sum_{v \in V} d = \frac{1}{|V|} |V| d = d$$

Problem 4

This problem is set in the context of of dynamic graphs, represented as a sequence of edge additions and/or edge deletions. The properties of interest refer typically to individual graph snapshots G_i at time i, therefore focusing on topological changes rather than on the whole graph evolution.

We consider a dynamic undirected graph $G(t, w) = (V, E(t, w))$ modeled by a sliding window of size w. We denote the number of nodes $|V| = n$ and the set of active edges $W = \{e_{t-w+1}, \ldots, e_t\}$. We consider the general case $t > w$ and $w \geq n$.
The algorithm we want to implement needs to account for the modifications in the graph structure induced by both the addition of a new edge and the deletion of the oldest edge at each time instant in W. Each edge added or deleted can impact connectivity in many different ways. We assume that the case of multiple edges cannot happen.

The main idea to address our problem is to keep track of connected components in the graph during the stream of edges, and deciding connectivity at each time instant by checking the current number of connected components. To achieve this, we need the primary data structures:

- A *disjoint-set* to store and manipulate connected components;
- A *data buffer* implementing a FIFO policy to store the active edges;
- An array containing all the vertices seen so far, representing the set V.

There are different data structures that could fit our needs.
A possible algorithm is described below.

When a new edge e_{t+1} arrives at time $t + 1$:

1. Enqueue e_{t+1} to the buffer of active nodes;
2. Discard the oldest edge in the buffer of active nodes;
3. Examine the endpoints of the new edge:

- If both of them are "new vertices", i.e. they aren't contained in V, add them as an independent connected component in the disjoint-set;
- If they are in two different connected components of the disjoint-set at time t, merge them;
- If one is in a connected component of the disjoint-set at time t and the other is "new", add it to the connected component of the first.

4. Because an edge was discarded potentially altering the graph topology, we need to recreate the disjoint set structure to identify the connected components after the edges update;

5. Check the number of connected components: if it's only one, the graph is connected, otherwise it is disconnected.

The space complexity of the algorithm depends on the structures we define and maintain: both the disjoint-set and the array for seen vertices require $\mathcal{O}(V)$ space, with V set of nodes currently in the graph. The buffer for active edges instead is $\mathcal{O}(w)$. The total complexity is then $2\mathcal{O}(V) + \mathcal{O}(w) = \mathcal{O}(V + w)$.

The update time between the graph snapshot at time t and the snapshot at time $t + 1$ derives from the computations needed to insert a new edge, discard the oldest edge and especially reconstructing the disjoint-set of connected components. The insertion and discarding operations take constant or nearly constant time, depending on the specific data structure we choose. Reconstructing components is computationally heavier. In the specific case of union-finds, it can be done in $\mathcal{O}(w + \alpha(V))$ time, where $\mathcal{O}(\alpha(V))$ is the complexity of a single union operation (expressed with inverse Ackermann function) and it needs to be repeated for all w active edges to build the sets of connected components.

Problem 5

In this problem we need to develop an opinion formation model that incorporates the idea of back-fire, that is the difference of opinions of two individuals tends to increase even more during the opinion formation process.
We propose a modification of the DeGroot weighted averaging opinion formation model. In this model, an individual's opinion is updated through an average of the individual's opinion and the opinion of neighbours. Let's recall its characteristics, given a graph $G = (V, E)$:

- Weight w on the edge that connects a node i to a node j, representing the trust of the first node to the second node;
- At time t, node i has opinion $x_i^{(t)}$ starting with $x_i^{(0)} \in [0, 1]$;

- Node i updates their opinion based on the formula:

$$x_i^{(t+1)} = \frac{w_{ii}x_i^{(t)} + \sum_{j|(i,j)\in E} w_{ij}x_j^{(t)}}{w_{ii} + \sum_{j|(i,j)\in E} w_{ij}}$$

Under conditions of G being strongly connected and aperiodic, the opinion formation process in a DeGroot model always converges.
Without changing the opinion updating rule, we can try and intervene on the weights. It's possible to define weights dynamically in a way that takes into account the level of agreement or disagreement between the individuals:

$$\tilde{w}_{ij}(t) = k_i x_i^{(t)} x_j^{(t)} + 1, \quad k_i \geq 0$$

For $k_i = 0$, this is equal to the DeGroot model.
The parameter k_i decides the amount of influence that the agreement/disagreement between nodes i and j will have on the weight w_{ij}. This way, the trust that a i has towards j also incorporates a component measuring the impact of difference/similarity of opinions. After redefining the weights, the opinions are updated in the same way as in the original DeGroot model:

$$x_i^{(t+1)} = \frac{\tilde{w}_{ii}x_i^{(t)} + \sum_{j|(i,j)\in E} \tilde{w}_{ij}x_j^{(t)}}{\tilde{w}_{ii} + \sum_{j|(i,j)\in E} \tilde{w}_{ij}}$$

As $k_i > 0$, the product $x_i^{(t)}x_j^{(t)}$ and decreases the influence of j on i, depending on the strength k_i. For negative weights w_{ij}, we have the back-fire phenomenon.

Problem 6

See the notebook `Problem6.ipynb` for the implementation.
This problem focuses on applying Graph Neural Networks for Node Classification and Link Prediction.

6.1 Exploratory Data Analysis

In this problem we utilize the PubMed dataset, a citation network with nodes representing scientific publications from the PubMed database, primarily in the biomedical field. Edges indicate citations between these publications. Training, validation and test splits are given by binary masks. Each node features a $TF-IDF$ weighted word vector from the publication's abstract from a dictionary which consists of 500 unique words and a class label that denotes the publication's subject category.
Let's take a better look at the dataset, shown in Figure 1.

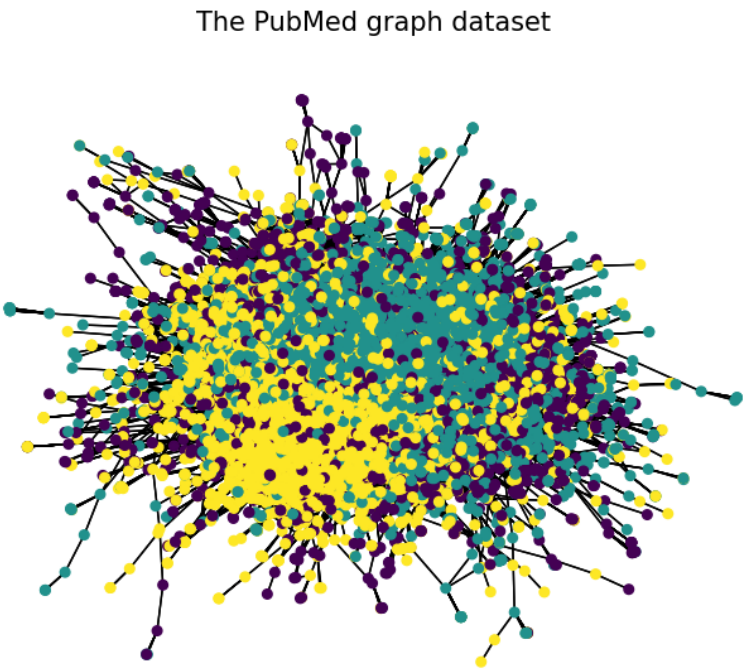


Figure 1: PubMed dataset visualization.

6.1.1 Network Structure

The PubMed dataset is structured as follows:

- Number of publications (nodes): 19717
- Number of citation links (edges): 88648
- Average citations per publication: 4.50
- Average clustering coefficient: 0.0602

We can observe the citation (i.e., the degree) distribution in Figure 2. The frequency on the y axis is expressed on a log scale for better visualization.
The dataset object contains:

- The feature matrix x, 19717 rows and 500 columns that represent the TF_IDF values associated to each node;
- The tensor `edge_index` containing the tuples of nodes connected by an edge;
- The vector of labels y, specifying a class label for each node;
- The train, validation and test datasets, provided in form of boolean masks.

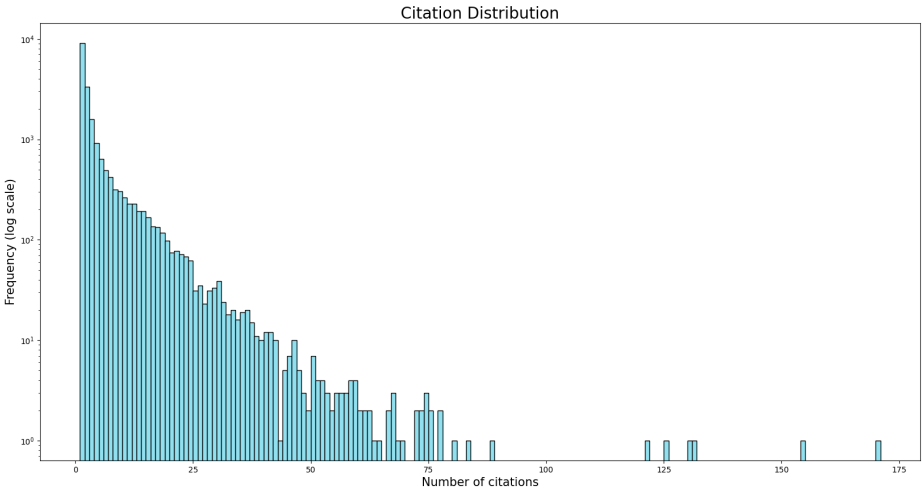


Figure 2: Citations distribution.

6.1.2 Network Features

Now we examine the spread of $TF-IDF$ values and explore potential correlations between word frequencies and publication categories.
First, let's recall that $TF-IDF$ combines two metrics:

- Term Frequency TF, measuring how frequently a term) appears in a document.

$$TF(t,d) = \frac{\text{Number of times term } t \text{ appears in document } d}{\text{Total number of terms in document } d}$$

- Inverse Document Frequency IDF, measuring how important a term is across a collection of documents.

$$IDF(t) = log(\frac{\text{Total number of documents}}{\text{Number of documents containing the term } t}) + 1$$

The $TF-IDF$ value for a term t in a document d is obtained by multiplying both quantities:

$$TF-IDF(t,d) = TF(t,d) \cdot IDF(t)$$

This value is proportional to the number of times that t appears in a document, but is also inversely proportional to the frequency of t in the collection of documents. This behaviour realizes the idea of giving more importance to the terms that do appear in the document, but are in general not too frequent.
Based the data we have, we don't have access to the vocabulary of terms for which the $TF-IDF$ has been computed, however we can still say something about it.
The TF_IDF has a minimum value of 0 and a maximum value of 1.2633. Within each document class, its distribution is shown in Figure 3.
Most values are contained in the interval $[0, 0.2]$.

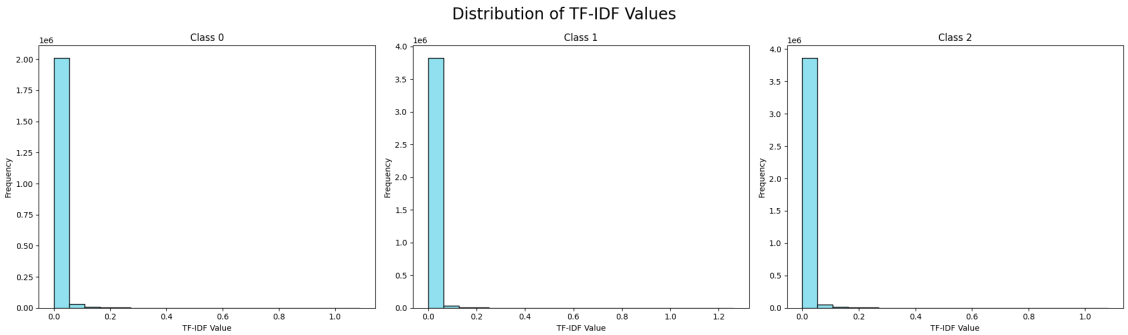


Figure 3: Distribution of TF_IDF across classes.

6.1.3 Class Distribution

We directly plot the class distribution in Figure 4 to have a clearer idea of the distribution of publications in the various categories:

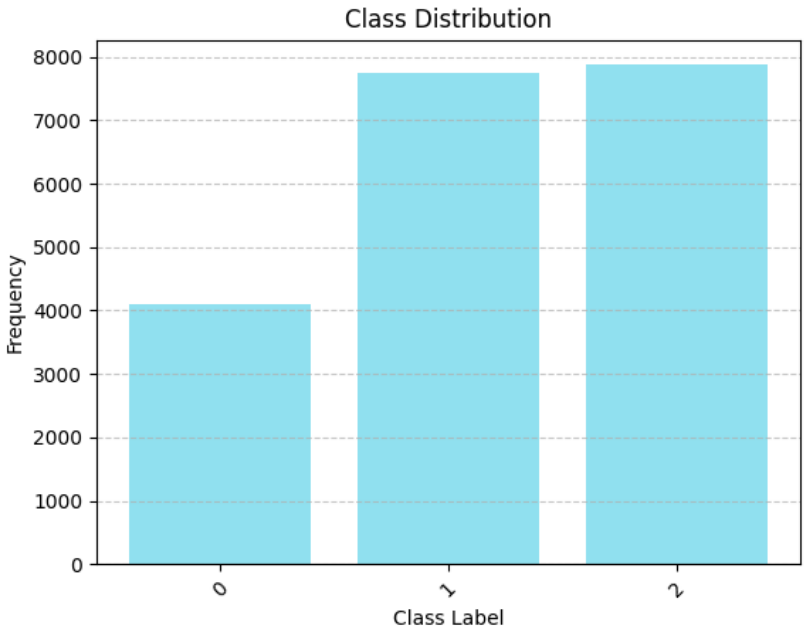


Figure 4: Distribution of class labels.

Class names are not directly available within the PubMed dataset object from the Planetoid provider. However, we know they are three and labeled with numbers $0, 1, 2$. Their frequency is showed in the above barplot. We can notice that, while class 1 and 2 contain almost the same number of documents (nodes), class 0 appears quite imbalanced with respect to the other two.

6.2 Node Classification

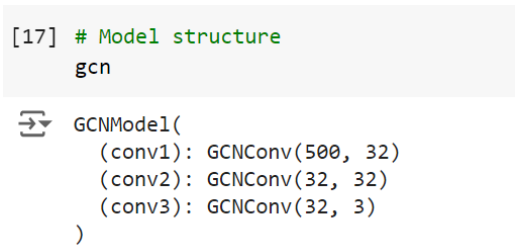
The first task we tackle is constructing and training a Graph Neural Network (GNN) to categorize publications within the PubMed dataset into their corresponding subject categories. In order to explore the characteristics and performances of different GNN architectures like GCN, GraphSAGE, and GAT, 4 different models were implemented.

1. **GCN Model**: the first model employs three Graph Convolutional Network layers, 32 hidden channels instead of the default 16 and the `leaky_relu` activation function.

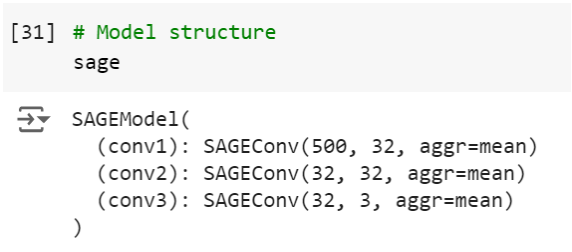
2. **SAGE Model**: the second model employs three SAGE Convolutional layers, a more sophisticated type of convolution that allows to capture more complex features. The hidden channels are again 32 and the activation function is again `leaky_relu`.

3. **GAT Model**: the third model employs two Graph Attention Network layers and one dropout layer, that helps making the attention mechanism more robust. The hidden channels are always 32, the attention heads are 8 and the dropout rate 0.5. The activation function used is `elu`.

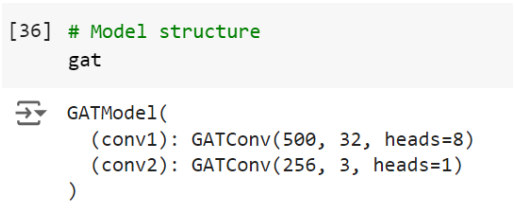
4. **Mix model**: the fourth model mixes layers from the previous ones. In particular, it uses one Sage layer, one GAT layer, another SAGE layer and a dropout layer. With the usual parameters for hidden channels, attention heads, dropout rate, one `leaky_relu` and one `relu` as activation functions.



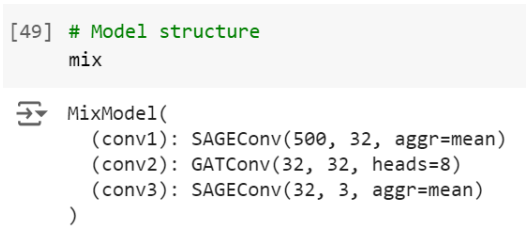
(a) *GCN model*



(b) *SAGE model*



(c) *GAT model*



(d) *Mix model*

Figure 5: Structure of the models used for classification.

The benchmark score to surpass is 80.2% accuracy.
The training loss and validation accuracy plots are reported at the following page, while classification performances are summarized in Table 1.

Model	**Test Accuracy**	**Precision**	**Recall**	**F1 Score**	**N. params**
GCN Model	87.37%	0.874	0.874	0.874	17187
SAGE Model	89.22%	0.893	0.892	0.892	34307
GAT Model	86.69%	0.867	0.867	0.867	129545
Mix Model	88.05%	0.881	0.881	0.880	40963

Table 1: Evaluation metrics for the different models

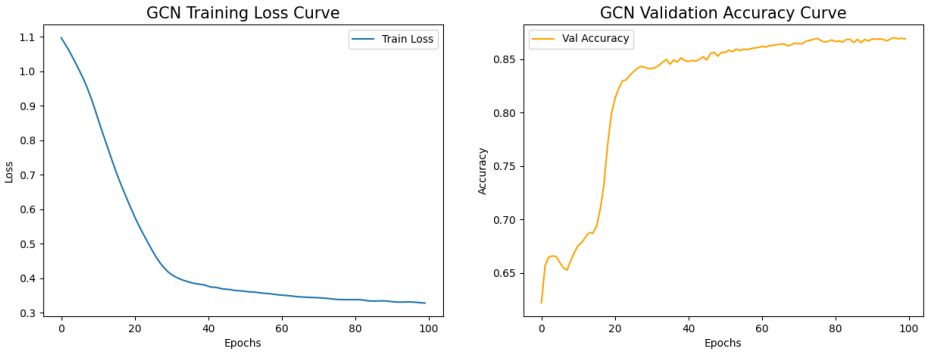


Figure 6: Training loss and validation accuracy of GCN model.

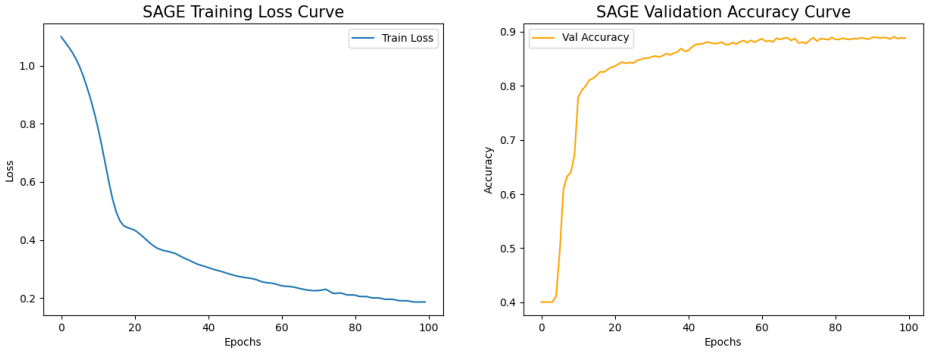


Figure 7: Training loss and validation accuracy of SAGE model.

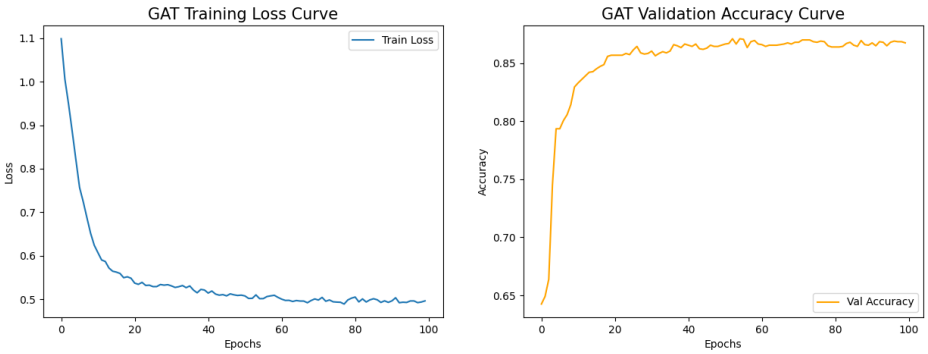


Figure 8: Training loss and validation accuracy of GAT model.

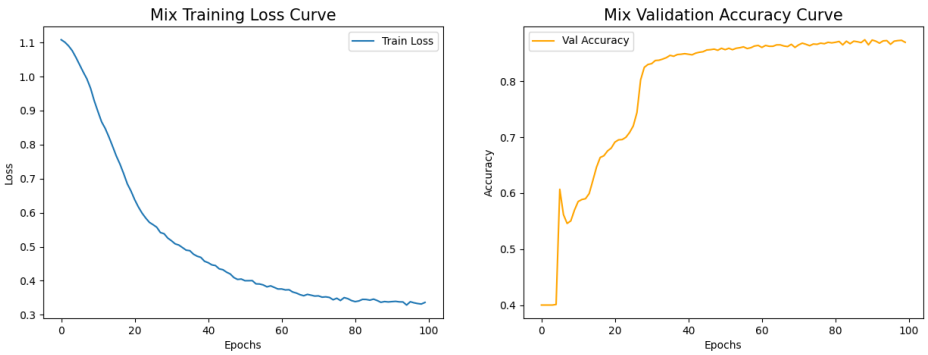


Figure 9: Training loss and validation accuracy of Mix model.

6.3 Link prediction

In this task we build 3 GNN-based models to predict the presence of citation links between publications, experimenting again with GCN, SAGE and GAT layers.

1. **GNN model 1**: this model features four GCN layers, 128 hidden channels, 64 output channels and a learning rate set to 0.005 to regularize the learning curve. The idea was to see how a more complex architecture would perform.

2. **GNN model 2**: this model is simpler, with just two GAT layers and a dropout in the encoding phase. Same number of hidden and output channels, the learning rate has the default value of 0.01. Maybe because of its smplicity, is the model that achieves lowest AUC score.

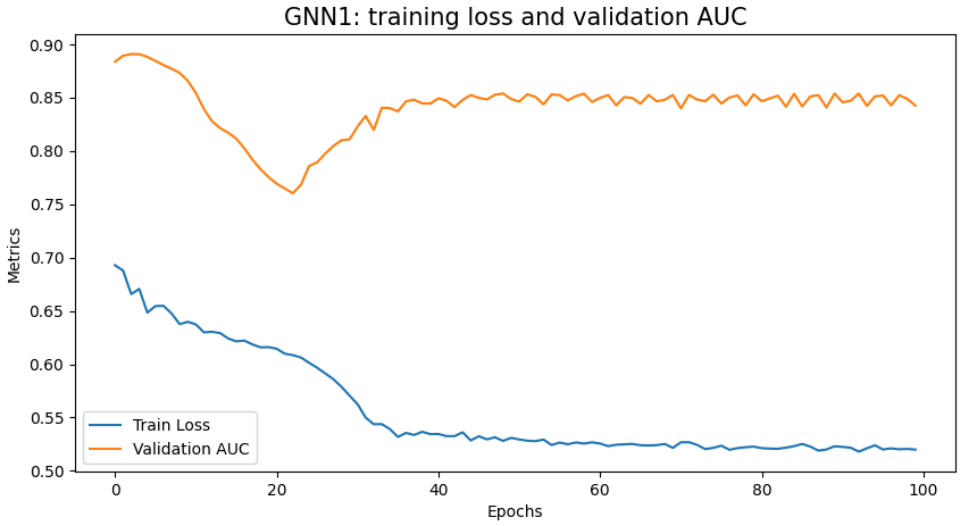
3. **GNN model 3**: this model is build with one GCN layer, two GAT layers and another GCN layer. The activation function is always `relu`. With standard training parameter values, is performs similarly to GNN model 1.

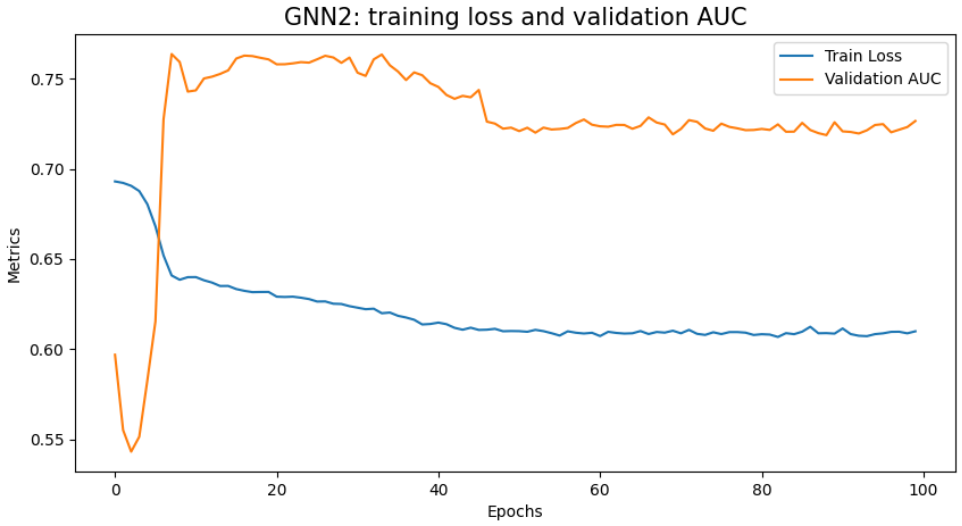
The benchmark score to surpass is 91.0% AUC.
The models' proficiency at distinguishing between existing and non-existing links is evaluated by means of metrics such as AUC-ROC, F1-score, and ranking metrics (in particular, MRR and Hits@K). The link prediction performances are summarized in Table 2.

Model	Test AUC	F1 Score	MRR	Hits@10	N. params
GNN Link Model 1	85.51%	0.7499	0.0107	10.0	105408
GNN Link Model 2	72.45%	0.6684	0.0369	10.0	72768
GNN Link Model 3	83.57%	0.7387	0.0104	10.0	105920

Table 2: Evaluation metrics for the different models

In Figure 10 are reported the plots of training loss and validation accuracy for each model.





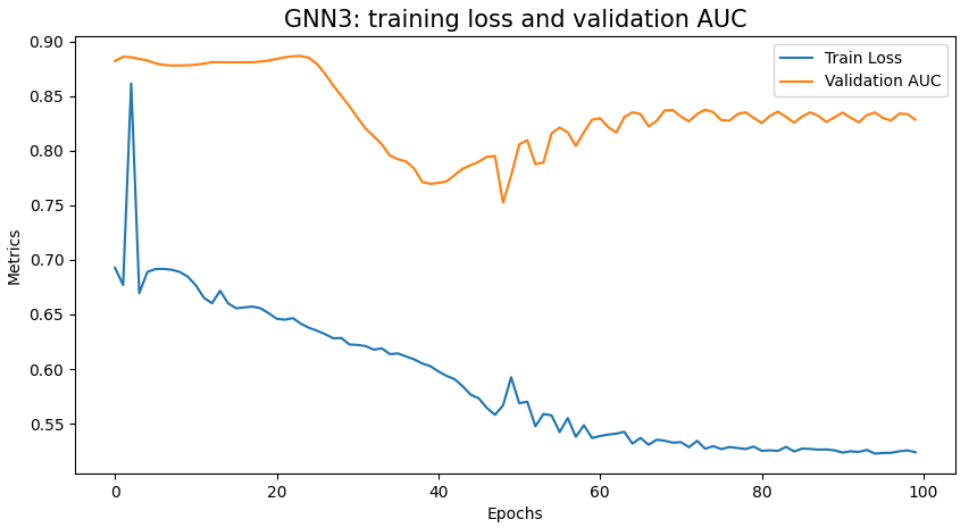


Figure 10: Training loss and validation AUC of the GNN models.

The main obstacle encountered in this problem has been the construction of the models. A brief overview of graph neural networks is certainly not enough to cover the plurality of possible architectures. Each type of convolutional layer comes with its own behaviour and characteristics, and finding the best combinations of architectures, hyperparameters, regularization mechanisms and convolution types to solve the assigned problems is difficult without a sound knowledge of the subject. Some aspects of learning can also be controlled during training phase: for example, tuning the learning rate, defining a number of epochs and implementing an early stopping criterion. This gives us a lot of freedom in choosing what should work best for us. The models implemented in this problem are some of the possible ones, but certainly far from being the best.