Hollywood Actors Community Detection And Genre Prediction

Sophia Barton
Stanford University
sophiapb@stanford.edu

Nidhi Manoj
Stanford University
nmanoj@stanford.edu

Flora Wang
Stanford University
fwang7@stanford.edu

Abstract

Data in many fields can be expressed in terms of networks, which offer a generic language for analyzing complex systems of interacting entities. Understanding the community structure of a network can illuminate the relation between graph function and topology, and assists in tasks such as node classification. We analyzed a network of Hollywood actors, in which actors are linked if they have co-starred in a movie. We implemented two community detection algorithms (Louvain and Clauset-Newman-Moore), as well as three Graph Neural Networks (GCN, GraphSage, GAT) in order to perform community detection on this dataset. Ultimately, the neural networks leverage network structure to perform node classification: we predict the genre of film that an actor is most likely to appear in.

1. Introduction

This paper focuses on community detection as a means to perform node classification on a dataset of Hollywood actors, which has nodes representing actors and edges encoding their co-acting relationships. Community detection has been a prominent field of interest in research understanding networks. Graph communities are defined as tightly connected sets of nodes, with many internal connections, or edges, and few external edges between the community and the nodes in the remaining portion of the graph.

The task of detecting clusters among Hollywood actors is interesting because the general public is familiar with actors, making the findings relatable. Analysis of clustering can identify similarities in terms of movie genres, acting styles, or even time period or country in which the movie was produced. Many applications of clustering analysis are possible, such as identifying actors/actresses who are similar to one's personal favorite actors/actresses, i.e. who act in the same genre of movies or produce similar kinds of content. Through identifying major co-acting relationships, one can also analyze graph neighbors and clusters to determine movie recommendations.

In addition to implementing two community algorithms - Louvain and Clauset-Newman-Moore - we experiment with Graph Neural Networks (GNN) in order to understand community structure, and perform node classification. In node classification, each node is associated with a single label, along with potentially other features which guide learning in addition to the graph structure, and the model predicts the label of nodes without using a ground-truth label.

2. Related Work

Community detection is a powerful and widely-studied topic, with many possible algorithms as solutions to this task - each with its own strengths and weaknesses. Graph Neural Networks utilize network structure such as communities in order to generate feature embeddings or labels, the latter of which is utilized in our prediction task.

2.1. Finding community structure in very large networks [1]

Clauset, Newman, and Moore optimize previous community detection algorithms that are computationally intensive and unsuitable for large networks. This is applicable to our project, given that the actor dataset we use has over 200,000 nodes and 6 million edges. Traditional community detection techniques maximize the modularity score. Modularity (Q) of a certain partitioning S of an unweighted graph is the sum over all partitions, of the number of edges within the partition minus the *expected* number of edges within the partition. Mathematically, modularity is:

$$Q = \frac{1}{2m} \sum_{i,j} \left[A_{ij} - \frac{k_i k_j}{2m} \right] \delta(c_i, c_j)$$

Note that $2m$ is the sum of all entries in the adjacency matrix; A_{ij} is the entry in the i^{th} row and j^{th} column of the adjacency matrix; $\delta(c_i, c_j)$ is 1 when i, j are in the same community and 0 otherwise. Communities are found by calculating which pair of nodes, i, j, has the largest ΔQ_{ij}. Clauset et al. optimize this approach by using more efficient data structures. Instead of maintaining the full adjacency matrix, they use a sparse matrix of ΔQ_{ij} only for community pairs i, j with at least one edge between them. The

optimized algorithm involves calculating the initial values of ΔQ_{ij} and a_{ij}, which is the fraction of edges that join nodes in community i to nodes in community j, populating a max-heap H with the largest element of each row in ΔQ. Then, until there is only one community left, the algorithm selects the largest ΔQ_{ij} from H, join the corresponding communities, update the matrix ΔQ, the heap H, and a_{ij}, and increment Q by ΔQ_{ij}.

Through these optimizations Clauset et al. were able to improve the run time to $O(md \log n)$, for a network of n vertices, m edges, and a depth of d in the dendrogram. In sparse and hierarchical networks, the time complexity is $O(n \log^2 n)$. One shortcoming of this method is that it can still be intensive with a large number of edges, m, and a large depth, d.

Because this method still greatly improves the performance of finding community structure in large networks, we will implement this approach on our data and use the results as a baseline for finding communities in our Hollywood actor network.

2.2. Graph Neural Networks: A Review of Methods and Applications [2]

This recently published paper offers a lengthy survey of Graph Neural Networks (GNNs), and introduces variants of this model, broken down by graph type, propagation type, and training type. First, it describes GNNs as connectionist models that capture the dependencies of graphs by using message passing between nodes. They keep a state which can represent information from a node's neighborhood with arbitrary depth. The idea behind GNNs has roots in CNNs, which can achieve expressive representations from localized spatial features. However, CNNs and RNNs are only functional on Euclidean data such as text and image, because they stack features of nodes by a specific order, but there's no natural ordering of nodes in a graph. The output of GNNs, in contrast, is the same regardless of the input order of nodes. The target of a GNN is to learn a state embedding or label which contains information about one's neighborhood, for each node, which can then be used to produce an output such as a node label. The vanilla GNN is composed of a local transition and local output function (both feed-forward neural nets) for each node, which are stacked to create one global transition function and one global output function. Each state is computed from the global transition function applied to the feature vectors, and the output from the previous round. One downside is that this iterative process of computing hidden states of nodes is inefficient. GNNs are typically used for supervised or semi-supervised learning.

The paper then discusses variants of GNNs for different types of graphs, such as directed, heterogeneous, or dynamic graphs, none of which describe our dataset. It then discusses variations of propagation types. Models can use different aggregators to gather information for each node's neighbors, and updaters to update the node's hidden state. One type of propagation is convolution, which uses both spectral and non-spectral approaches. The authors explain GCNs, or graph convolution networks, which aims to fix the problem of overfitting on local neighborhood structures. This model works well for graphs with wide node degree distributions, which applies to our dataset. Another propagation method includes gates, which are used to improve long-term propagation of information across the graph structure. For example, GGNNs, or gated graph neural nets, use a GRU or LSTM in the propagation step. Furthermore, attention is another variant. An example model is GAT, or graph attention network, which attends to neighbors when computing a node's hidden state.

A drawback of the original GCN is that it requires the full graph Laplacian, and the embedding of each node at layer *L* is computed from the embeddings of all neighbors at layer *L* - 1, which means the growth is exponential so it's difficult to compute the gradient for even just one node. Thus, other training methods include sampling to replace the full Laplacian with learnable aggregation functions, and in the case of Stochastic Fixed-Point Gradient Descent for GNNs, alternating between sampling nodes to update embeddings and sampling labeled nodes to update parameters. Mixtures of model variants are also described. For example, MPNN, or message passing neural net, is a mix of GNN and GCN approaches. The paper also goes into some detail describing Graph Networks, and the GN block composed of three udpate functions and three aggregation functions, and allows for flexible representations of different attributes and graph structures; an arbitrary number of GNs blocks can be stacked in sequence with shared or unshared parameters.

This paper relates to the material that we covered this week in this course when we started discussing Graph Neural Networks, and next week when we learn this topic further as well as Deep Generative Models for Graphs, since this survey also briefly discusses applications of GNNs for generative models. This paper relates to the other two works we review here because of the potential to apply a GNN model - any variant - to the task of community detection.

A strength of this paper is that it provides a very exhaustive list of many variations that are possible using the general GNN framework, and has a clear layout in the format of discussing the motivation behind the creation and use of GNNs, variants for different graph structures, propagation types, training methods, and applications. It also does a good job of giving rigorous explanations and equations for most the models it describes. This is helpful for readers who may choose to implement one of them. A weakness of this paper, though, is that it often just lists several models within a few sentences, and doesn't elaborate how they are related

to the current discussion, their importance, or any details of how they work. Thus, since this claims to not be an exhaustive list of all models and applications of GNNs, it is not optimal to just list many models without detail. Instead, it would be better to either explain them in a bit more detail, or to remove them from the paper and only keep them in the summary graphic.

Further research questions that the paper mentions include discerning how to construct deep GCNs, because at the moment they can not stack too many on top of each other because this causes over-smoothing in which nodes converge to the same value. Another question is how to scale GNNs since many steps are computationally consuming. For our work, we investigate one of the methods listed for unsupervised learning - the GCN - which is applicable to graphs with wide node degree distributions.

2.3. Graph Neural Networks for community detection with modularity-based attention model [3]

While the first paper discusses community detection and the second paper delves into the details of Graph Neural Networks, this recently published paper by Lobov and Ivanov brings together the concepts of community detection and Graph Neural Network. A lot of the technical concepts from the other discussed papers are seen in this paper including modularity, as well as GNN implementations. This paper focuses specifically on clustering of nodes which are not labeled with communities.

Our course discussed various Graph Neural Network approaches. In terms of novelty and added contribution, this paper proposes an approach that involves using an unsupervised GNN with attention through a relaxed modularity for soft labelling which is differentiable and makes learning possible. Lobov and Ivanov also show that this approach is comparable to supervised neural networks which produce state-of-the-art performance but require a lot of parameters and computation time.

We learned about modularity and log-likelihood optimization in class and in our first assignment. This paper proposes a form of modularity where for each node, the probability of being in each cluster is calculated. The paper then shows that this enables differentiability for a neural network model and thus enables us to work on cluster detection using GNNs.

Regarding attention, the authors explain that they were inspired by attention mechanisms in transformers (which are frequently used in NLP problems, see [4]) and used a similar encoder structure to produce embeddings and to compute the likelihood that a node is in each cluster. Seeing the improvements that attention introduces in this paper, we also experiment with attention in our GNN implementations in order to improve the community detection and node classification performance.

While this paper brings together the concepts of clustering and GNN and demonstrates performance comparable to state-of-the-art supervised methods, the paper can be improved in some ways.

The related work section in the paper is very short and brushes over much of the previous work that preceded their work. The authors explain that their model is an improvement on previous work given their employment of attention and optimization of a soft modularity loss function. However, they only have one sentence mentioning previous work in GNN with attention mechanisms, and they only say that the previous work had prominent results in other fields. Identifying previous work is important because it provides context for readers and gives credit to those who have come before. Similar work had in fact been done in NLP, speech recognition, and other fields so a thorough explanation would have been more ideal than the brief sentence and two citations. While the new improvements and their approach is explained thoroughly, more context about attention mechanisms would have been helpful and made the paper more thorough. The authors also assumed that the audience has a strong understanding of concepts such as Stochastic Block Model, Graph Neural Networks, and why they decided to use the strategies they did. Nevertheless, the paper provides a lot of interesting technical insights.

3. Data

We are using a Kaggle dataset [1] of 45,000 movies and information of the actors in the cast of each movie. We generated the network of co-acting relationships ourselves. The nodes in the graph are Hollywood actors and the edges represent co-acting relationships; in other words, an edge exists between two nodes in the graph if the two actors have worked together in at least one movie. The dataset also has metadata on the genre of the movies the actors have acted in. We extracted the most common genre associated with each actor. The original dataset yielded a graph of 202,747 nodes and 6,183,763 edges, but after data cleaning, the network we generated contains 199,710 nodes and 6,156,707 edges since we opted to only including actors which have a most common genre. There are 20 possible 'top genres.'

3.1. Data Analysis

In the full graph we created, there are 202,379 nodes and 6,183,763 edges, making this a massive dataset. Further analysis of the data shows that the largest component within this contains 196,987 nodes, indicating that a large majority of the nodes are connected to each other.

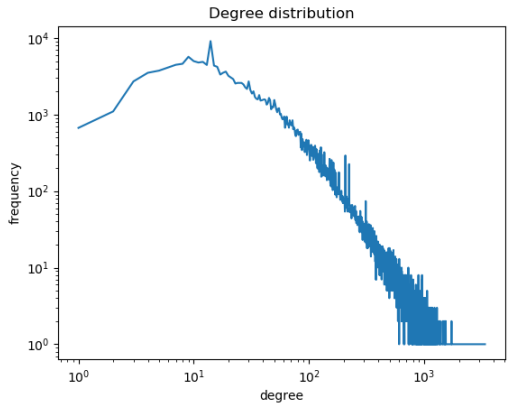
The average degree is 61.11, which shows that most actors in the dataset have worked with a large number of other

[1] https://www.kaggle.com/rounakbanik/the-movies-dataset

actors and that there are at least some dense parts of the networks.

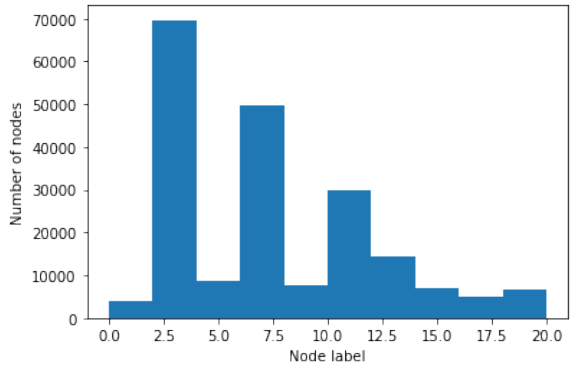
The following graph shows the distribution of degrees in the Hollywood Actor network.

Figure 1. Distribution of degrees in the Hollywood actor network



We also generated labels for the actors based on the most common genre of films that they've starred in. We did this by finding the genres associated with the movies that each actor has acted in and assigning the genre with with most movies to the actor as its "label," used later in our GNNs models for node classification. The following chart shows the distribution of labels in the graph of 199,710 nodes and 6,156,707 edges (the graph excluding nodes that don't have an associated 'top genre'). The most common label (genre) that is associated with the actors in our network is "Family" (label = 2).

Figure 2. Distribution of most common genre in the Hollywood actor network



Here is the mapping of the 20 top genre names to their label value: { 'Action': 0, 'War': 1, 'Western': 2, 'Crime': 3, 'Horror': 4, 'History': 5, 'Romance': 6, 'Fantasy': 7, 'Science Fiction': 8, 'TV Movie': 9, 'Family': 10, 'Music': 11, 'Adventure': 12, 'Thriller': 13, 'Drama': 14, 'Comedy': 15, 'Animation': 16, 'Foreign': 17, 'Mystery': 18, 'Documentary': 19, 'MISC': 20 }.

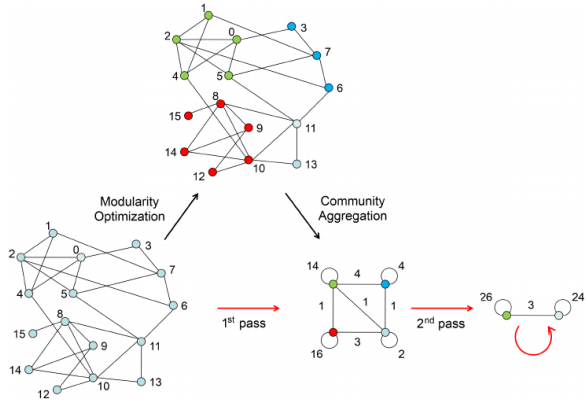
Given the massive size of the dataset, we conduct our baseline algorithms on only a subset (100,000 edges) of the graph which contains actors that have a top genre, in order to increase the speed of our models.

4. Algorithms

4.1. Baseline: Louvain, Clauset-Newman-Moore

Our baseline model performs community detection without a GNN. We first attempted coding the Louvain algorithm from scratch (using the pseudocode from lecture 4). Then we also utilized an API from PyPi (Python Package Index) which outputs the partitioning of the nodes which maximizes the modularity score, using the Louvain algorithm. Furthermore, we implemented a slightly more complicated method for community detection using the optimized Clauset-Newman-Moore algorithm discussed in [1].

Figure 3. Louvain algorithm overview



4.2. Graph Neural Network

Given that the dataset is very large and has other metadata information, we experiment with the deep learning approach of Graph Neural Networks (GNNs) to improve our performance with community detection and node classification. GNNs aim to generate node embeddings by iteratively aggregating feature information from local graph neighborhoods; embeddings can then be used for classification. We want to classify each node in our Hollywood actor network with a cluster assignment where the communities are based on the genre of movie that the actors have acted the most in. We include the genre the actor has acted the most in as the node labels for the GNN to perform node classification.

Since we wanted to use our own data, not data already processed and available in the libraries, for the GNNs, a large portion of our time was used to process the data and transform the data to an appropriate format in Pytorch that can be fed into our models. Initially, we attempted to implement an InMemoryDataset class for our data, similar to the class Planetoid, which is used to load the commonly used

Cora dataset. However, we were able to just create a Pytorch Data object with our network edges and node labels to feed into our GNN models.

We implement three types of Graph Neural Networks: Graph Convolutional Network, GraphSage, and Graph Attention Network. We included the same node labels and the edges for each GNN and ran each GNN for 200 epochs. We used a learning rate of 0.01, a dropout value of 0.5, and 2 layers for all of the GNNs.

4.2.1 Graph Neural Networks

In general, graph neural networks have four components:

1. Message computation: Neural networks learn a message function between nodes, which can be expressed as $M(h_u^k, h_v^k, e_{u,v})$ for a pair of nodes *u* and *v*; h_u^k refers to the hidden representation of node u at layer k, and $e_{u,v}$ is any information such as edge weight or other features.

2. Aggregation: Each layer applies a function to aggregate information from all neighbors of each node. The function is typically permutation invariant so that ordering of neighbors does not affect the calculation.

3. Update: The representation of a node is updated based upon the aggregated representation of its neighborhood.

4. Pooling: This phase is only relevant to pool all individual node's representations into one representation of the entire graph. Popular options include mean, max, and sum.

4.2.2 Graph Convolutional Network

The idea behind GCNs is that each node's neighborhood defines a computation graph, and the model can learn to propagate information through the graph to compute node features or labels. Nodes aggregate information (features, labels, or embeddings) from their neighbors using neural networks. Models can be of any depth, and nodes have embeddings at each layer. The basic approach is to average neighbor messages and apply a neural network. The initial 0th-layer embeddings are just node features (or labels). Then, the embedding after *K* layers of neighborhood aggregation are computed as:

$$\mathbf{h}_v^k = \sigma\left(\mathbf{W}_k \sum_{u \in N(v)} \frac{\mathbf{h}_u^{k-1}}{|N(v)|} + \mathbf{B}_k \mathbf{h}_v^{k-1}\right), \forall k \in \{1, ..., K\}$$

which is a non-linearity (such as ReLU) applied to a weighted average of the neighbors' previous layer embeddings, and previous layer embedding of node v itself.

The message computation, aggregation, and update steps can be expressed as a layer-wise propagation rule:

$$h^{k+1} = \sigma\left(D^{-\frac{1}{2}} A D^{-\frac{1}{2}} h^k W^k\right)$$

where h^k represents the matrix of activations in the *k*-th layer, $D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$ is the normalized adjacency of the graph, and W^k is a layer-specific learnable matrix.

The aggregation phase of a GCN uses a weighted sum, where the weight for aggregating from v to u is the (u, v) entry of the normalized adjacency matrix ($D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$). The update phase is a multi-layer perceptron (MLP).

Generated embeddings can be fed into any loss function and stochastic gradient descent can be used to train the weight parameters. Unsupervised training means that the GCN model only uses the graph structure to generate embeddings or labels. In our case, we did not use features but predicted labels of the most likely genre (ie 'top genre') of actor nodes.

4.2.3 GraphSAGE

Both GCN and GraphSAGE employ similar transfer of information and message function where **W** and **b** are the weights and bias term of a neural network linear layer, and the neighbors of a node are simply defined as nodes that are connected to the node. The key difference is in the update of node representation which is based on the aggregated representation of the neighborhood. While GCNs utilize a multilayer perceptron (MLP), GraphSAGE uses a skip layer with the perceptron. Through concatenation, GraphSAGE employs a skip connection between the various layers of the algorithm, which contributes to performance gains. In the pooling layer (which we did not implement), each neighbor's vector would be passed in a fully-connected neural network followed by an element-wise max-pooling in order to aggregate information across the neighbors.

4.2.4 Graph Attention Network (GAT)

GAT improves upon the simple neighborhood aggregation methods (used in GCNs) by applying attention coefficients (weights), α_{vu} to node features. The GAT applies arbitrary importances to different neighbors of each node in the graph. The flexibility of applying different weights to different neighbors can be learned by the model. The attention mechanism of GAT is also beneficial because it is computationally efficient (the computation of attention coefficients can be parallelized across the edges in the graph) and storage efficient (there is a fixed number of parameters).

Our GAT model calculates attention coefficients in our message function and normalizes these coefficients with a softmax function. We only use one head to simplify our

implementation. However, future iterations of this analysis could use multi-head attention in order to stabilize the learning process. [5]

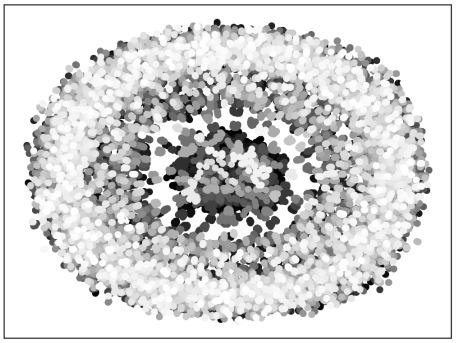
5. Results

5.1. Baseline

We first implemented the Louvain algorithm from scratch. However, this was not able to handle the size of our dataset and run at a reasonable time, given our lack of optimization.

Thus, we resorted to using a pre-existing implementation of this algorithm. Our results from plotting the partitioning of graphs, which was computed using the Louvain algorithm from PyPi, are displayed in Figure 4.

Figure 4. Communities found from the Louvain algorithm

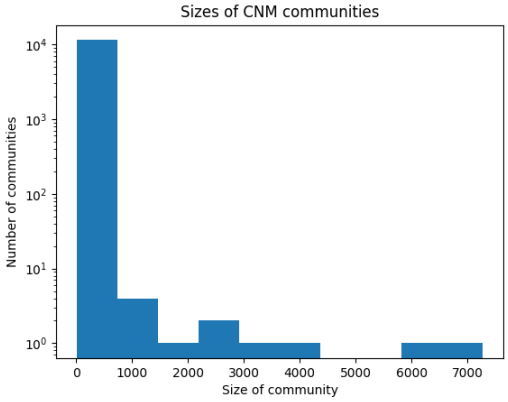


From the image, we can see that there is a densely connected component that is a ring, which contains most of the actors in the dataset. There is also a smaller densely connected component in the center of the image, which maintains relatively few connections to the outer ring. It is apparent that for any randomly selected node, it has a high probability of being connected to most other actors through a short number of connections, highlighting the fact that many actors have an impressive resume of working with many other actors.

We also ran an even-further optimized version of community detection: the Clauset-Newman-Moore (CNM) algorithm. We attempted to run the CNM algorithm on the full dataset. However, we ran into run-time errors, even with 102 GB of memory on Google Cloud. Thus, we took a subset of our data and ran CNM community detection on 100,000 edges (87,676 nodes). The average degree of this subset of data was 2.281. There were 11,237 connected components, and the largest connected component had 58,801 nodes. The CNM algorithm on this subset of data found 11,431 communities, which resulted in a modularity score of 0.841188.

Figure 5 shows a histogram of the sizes of each community resulting from this Clauset-Newman-Moore algorithm. The sizes of the community range from a minimum of 2 nodes to a maximum of 7426 nodes.

Figure 5. Histogram of the sizes of the communities found from the Clauset-Newman-Moore algorithm



Some of the actor communities we found via CNM included:

- [Anthony Mondal, Michael Cline]
  This makes sense since both Anthony Mondal and Michael Cline acted in the same movie: Sabrina (1995)

- [Dina Meyer, Warren Sulatycky, Coyote Shivers, Lynne Adams, Glenn Bang, Javon Barnwell, Sonny Marinelli, Beverly Murray, Bronwen Booth]
  This community includes several actors based in Canada and most of these actors have acted in TV shows.

- [Denis Sandler, Antoni Petorozliev]
  These two actors starred in a French documentary, Victor Young Perez (2013), but not in many other movies. Denis Sandler has been in 5 films. Antoni Petorozliev has only been in 1 movie.

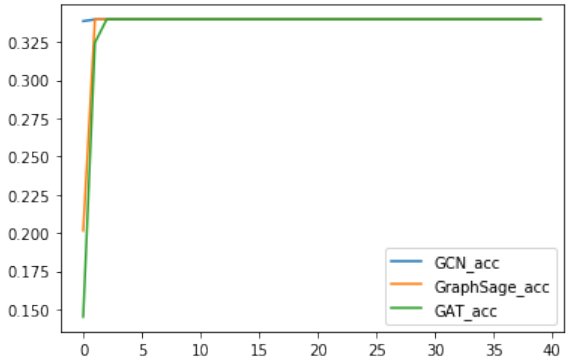
Based on this brief qualitative analysis of three random communities, it appears that many of the smaller communities (on the order of 10^4) that the CNM algorithm generated included actors that have worked in fewer and smaller films.

5.2. Graph Neural Networks

Figure 6 shows the accuracies of our three GNN models throughout the training process. The accuracies stayed very consistent across the three models with a maximum value of approximately 0.33. The reason why the accuracies stayed consistent across the epochs and did not improve much is likely because we did not have node features associated with the nodes. The GNN models use node features

in neighborhood aggregation. Furthermore, GAT offers improvements to the basic GCN model by applying attention coefficients to node features. Thus, without node features, the GAT model is unable to perform better than the GCN model. With only the network structure (edge indices) and the node labels, the GNNs were able to do some level of prediction but were ultimately limited in their performance.

Figure 6. Accuracies of Graph Neural Network Models



6. Conclusion and Future Work

The Louvain algorithm found that most actor nodes can be classified into one large connected component with the majority of actors in one community. Some actors (in the center of the Figure 4) are highly connected and represent actors that co-acted in many movies and with various other actors.

Our results from Clauset-Newman-Moore community detection show 11,431 communities with a modularity score of 0.841188 in the subset of our co-actor network dataset. The large connected component with 7426 nodes shows that a large majority of actors are part of one community, connected by movies made by the major film studios. The smaller communities include communities of actors in smaller films, who also do not act in many movies. The large number of communities produced by CNM shows some of the drawbacks of the CNM model. It is difficult to analyze why all of these communities are generated.

The results from our GNN models were lower than expected. Even though we included data on network edges and node labels, we did not have node features readily available for our actors. However, this mechanism and approach of using GNNs is relatively new and has not been applied to many contexts. GNNs have been primarily used for node classification and link prediction. We use GNNs for node classification as a way of detecting communities based upon network structure and most common genre labels, after a lot of research and discussion with TAs. Even though the models did not perform as well as we would have hoped, we think that our approach in generating a compatible Pytorch data object with new data and running the GNN on our actor network is still a compelling contribution. We processed the data, assigned labels, and constructed the network all from scratch. We additionally wrote the data loading from scratch so that we could feed in the dataset effectively into the Baseline (Louvain and lauset-Newman-Moore) and Graph Neural Network Models (GCN, GraphSAGE, GAT).

We have applied a myriad of community detection algorithms to our dataset of Hollywood actors (co-acting relationships and common movie genres they have worked in). However, there are several ways to improve upon what we have done. If we had more time, we would explore the following future steps.

1. Including further node features for the three GNN models. We have a lot of information available in our dataset including revenue made by movies, cast and crew information, etc. We only utilize the genre that the actor is most involved in. Further possible features for actor nodes could include number of movies done, average revenue, demographics (race, gender, age).

2. Tweaking the hyperparameters, including learning rate, dropout, number of layers, number of heads (GAT), of the GNN models. We experimented with a lot of models and while this surfaced a lot of great insights we would love to tune each model further.

7. Acknowledgements and Contributions

We would like to thank Jure Leskovec and Michele Catasta as well as the CS224W course staff for their guidance and support with this project.

We all contributed equally to the project by co-programming the code and model implementations and co-writing all the papers.

8. Code

Our code and data used for this analysis can be found at:
https://github.com/nidhimanoj4/CS224W_Hollywood_Graph_Networks

References

[1] A. Clauset, M. E. J. Newman, and C. Moore, "Finding community structure in very large networks," *Physical Review E*, vol. 70, Dec 2004.

[2] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, and M. Sun, "Graph neural networks: A review of methods and applications," *CoRR*, vol. abs/1812.08434, 2018.

[3] I. Lobov and S. Ivanov, "Unsupervised community detection with modularity-based attention model," *CoRR*, vol. abs/1905.10350, 2019.

[4] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *CoRR*, vol. abs/1706.03762, 2017.

[5] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," 2017.