Issues in Information Systems
Volume 25, Issue 4, pp. 344-360, 2024

DOI: https://doi.org/10.48009/4_iis_2024_128

Book structure and genre classification using GNNs

Rebecca C. Williams, *Georgia Southern University, rw06481@georgiasouthern.edu*
Jongyeop Kim, *Georgia Southern University, Jongyeopkim@geogriasouthern.edu*
Yao Xu, *Georgia Southern University, yxu@georgiasouthern.edu*

Abstract

Graph Attention Networks (GATs) are a specific form of Graph Neural Networks (GNNs) that aggregate hidden features from nodes and use self-attention mechanisms to re-weight edge values. This enables GATs to effectively discern the significance of various relationships within the graph. In this study, we leveraged these characteristics of GATs to convert text-based books into graph networks, train the model, and classify the genre of the books. Analyzing the performance of GATs, we explored a novel approach that converts text sentences into graph notation. By transforming text-based books into graph structures, we leveraged the strengths of GATs to classify the genres of the books. These datasets comprised book graphs, with entities represented as nodes and sentiment analysis scores of the relationships between these entities represented as edges. The datasets varied in graph size and construction, and we applied the same GAT model configurations across all tests to measure accuracy. The experimental results demonstrated the potential of GATs for successful book genre classification and provided valuable insights into how the size of input graphs affects GAT performance.

Keywords: GNN, GAT, Graph-Level Classification.

Introduction

Graph Attention Networks (GATs) are a specific form of Graph Neural Networks (GNNs) designed to aggregate hidden features from nodes and use self-attention mechanisms to re-weight edge values(Veličković, P. 2018, Kipf, T. N., & Welling, M. 2017). This enables GATs to effectively discern the significance of various relationships within a graph. In this study, we leveraged these characteristics of GATs to convert text-based books into graph networks, train the model, and classify the genre of the books. Traditional text classification methods often rely on linear or sequential processing of text, which may overlook the complex interdependencies between different elements within the text.

In contrast, GNNs offer a more sophisticated approach by representing the text as a graph, where nodes represent entities such as characters, locations, or significant terms, and edges represent the relationships between these entities, quantified through sentiment analysis scores. This graph-based representation allows for a more holistic analysis of the text, capturing intricate relationships that are otherwise missed by conventional methods (Shen , Y., et al., Cheng,2020). We explored a novel approach that converts book text sentences into graph notation. By transforming text-based books into graph structures, we utilized the strengths of GATs to classify the genres of the books, exploring a novel transformation method utilizing a pruning algorithm.These datasets varied in graph size and construction, however we applied the same GAT

model configurations across all tests to measure accuracy. This consistent application enabled us to evaluate the robustness and scalability of GATs across different graph structures.

The experimental results demonstrated the potential of GATs for successful book genre classification and provided valuable insights into how the size of input graphs affects GAT performance. Larger and more complex graphs, representing richer and more intricate narratives, presented unique challenges and opportunities for the GAT model. Our findings indicated that while GATs are highly effective at capturing and utilizing the relationships within smaller, simpler graphs, they also scale well to larger, more complex structures, maintaining high classification accuracy. By exploring the interplay between graph size and GAT performance, this research contributes to a deeper understanding of how graph-based representations can enhance classification tasks.

Specifically, our study sheds light on the scalability of GATs and their ability to handle varying levels of complexity in text data. This understanding is crucial for future applications of GATs in more extensive and diverse datasets, beyond book genre classification. In the following sections, we will delve into the theoretical underpinnings of Graph Attention Networks, describe our methodology for converting text into graph notation, and present the experimental setup and results. Through this exploration, we aim to highlight the efficacy of GATs in the domain of book genre classification and pave the way for future research in applying graph-based models to various classification tasks. By advancing the methods of text representation and classification, we hope to contribute to the broader field of machine learning and its applications in data-rich environments.

Background

The nature of GNNs connects multiple scientific and technical disciplines. In order to understand how GNNs work a basic understanding of Neural Networks (NNs) is necessary.

Neural Networks: The University of Stanford's Computer Science Department traces the history of NNs all the way back to the ongoing neuro research of the 1940s (Clabaugh, C., et al., 2000). During this time neurons were modeled with their behavior translated using electrical circuits. During the late 50's, Stanford used simulated hypothetical NNs for the first time against a real-world problem. Soon after in 1962, those same developers (Widrow and Hoff) implemented learning using weights. This simple idea would evolve into back-propagation which learns slowly utilizing a model structure composed of multiple layers.

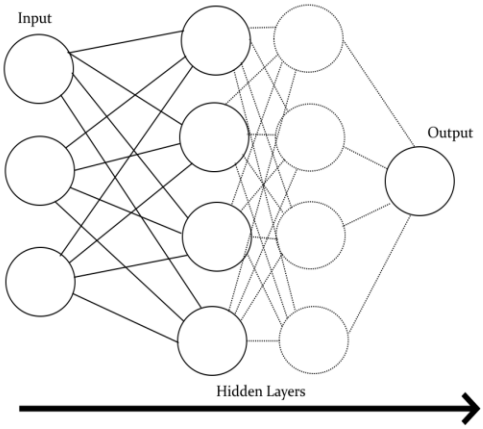


Figure 1: An example of NN architecture with data from input nodes flowing into a series of hidden layers eventually leading to a single output node which would contain the predicted value.

Weight represents how strong connections are between neurons. During forward propagation, this value is used by hidden layer neurons to process inputs. For a given hidden layer neuron, all connected input neurons (visualized in figure 1) are multiplied against their weight before being summed together and sent through an activation function (Ghorakavi, V., 2024, et al., January 3). After a prediction is generated, it is compared against its true value. The model then uses the amount of calculated error to change the neuronal weights (backpropagation). This entire process of forward followed by backpropagation is completed multiple times until prediction error is satisfactorily low.

Graph Neural Networks: The graph domain is large and sprawling. A graph itself showcases not only data (nodes), but also relationships between datums (edges). Both nodes and edges can even hold additional information. Historically, according to the paper, Graph neural networks: A review of methods and applications (Zhou, J., et al.,2020), GNNs can trace their origins back to Recursive NNs first introduced in the 90s. Recursive NNs (RvNNs), work with recursive data structures and are particularly useful when it comes to hierarchical data. Their structure allows them to process the relationships between parent and child nodes by pooling child node information together to form representative versions of the parent nodes(Bhakta et al.,,2024, January 6).

Recurrent Neural Networks or RNNs were introduced in 2009 (Zhou, J., et al., 2020). RNNs work primarily with sequential data, where predicted outputs are not necessarily independent from one another. In other words RNNs allow for the use of context, with hidden internal states acting as memory. The cyclical nature of using old memories to find new information makes RNNs well suited to handle problems involving cycles. In 2015 (Zhou, J., et al., 2020), advancements in Convolutional NNs (CNNs) led to the true graph representation ability that is foundational to GNNs today.

CNNs are mainly used with images. They feature Convolutional Layers, which break up the image into smaller sub images (using kernels) multiplying the kernel weight against the corresponding sub image, and pooling layer which reduces the size of feature data by combining feature values, usually taking the max or using the average(GeeksforGeeks, 2024). This idea of pooling features in a region of data into a feature that can represent the entire region sits at the core of how GNNs function.

Instead of a pooling layer, GNNs utilize message passing. Essentially node states are updated based on information pooled or aggregated from its neighbors. The same process can occur between the edges of a graph. A so-called context vector can even be used to store global pools of data(Sanchez-Lengeling, et al., 2021). Much like NNs, GNNs are very versatile and can be used for node, edge, and graph level prediction tasks. Node level tasks involve the pooling of nodes, edge level tasks involve the pooling of edges, and graph level tasks involve the context vector and global pooling. There are also different GNN models.

Graph Attention Networks: Graph Attention Networks (GATs) were first introduced in 2018, in the paper Graph Attention Networks(Veličković, et al.,2018). This novel GNN architecture features a graph attention layer which calculates the importance of one node's feature to another agnostic to the actual structure of a graph, though this process is artificially limited to only the neighborhood of the given node. In (Brody, S., Alon, et al., 2022) researchers introduce GATv2. This version served to improve a shortcoming of the original GAT whose attention function statically and globally ranked nodes. GATv2 allows for dynamic attention meaning node ranks change per given node. This showed a remarkable improvement in accuracy.

Related Work

BertGCN: The paper, Transductive Text Classification by Combining GCN and Bert(Lin, Y, 2021), details a proposed GCN (Graph Convolutional Network) model that constructs a graph in which nodes represent full document texts, or full documents themselves, and node features are pretrained BERT representations. According to Cameron Hashemi-Pour and Ben Lutkevich, site editors for TechTarget.com (Hashemi-Pour, C., Lutkevich, B, 2024), BERT or Bidirectional Encoder Representations from Transformers, is a NLP machine learning framework that is able to determine text meaning utilizing context provided by the surrounding text, as opposed to analyzing the text sequentially. BertGCN leverages Bert and the GCN model for use in text classification tasks.

The BertGCN model based on the previous TextGCN (Yao, L., et al., 2019) model also includes frequency-based edges within the input graphs. Accuracy-wise it improved on TextGCN, with mean accuracies of 89.3, 98.1, 96.6, 72.8, 86.0, for its datasets 20NG, R8, R52, Ohsumed, and MR respectively. The main difference between the TextGCN and BertGCN being that TextGCN uses an identity matrix, representing word and document nodes, for its node features, whereas BertGCN uses the Bert model.

Text Level Graph Neural Network: The paper (Huang, L, et al.,2019), in contrast to BertGCN which utilizes a combined corpus-level graph for its input, instead uses an input graph individual to its texts. In each text-level graph, word nodes are connected based on a small window, with edges representing words close to each other (within the window) in the text. The Text-Level GNN also utilizes message passing for convolution using the MPM method which updates node representations based on an aggregate of adjacent node information and original node representations.

Originally introduced in the paper Neural Message Passing for Quantum Chemistry by (Gilmer, J., et al.,2017) the MPM method used is one of various variations of MPNNs introduced by Gilmer and their team. In fact, the Gilmer paper describes commonly used MPNNs of its time, exploring novel variations leading to the development of general MPNN methods along with improvements on several different MPNNs. These improvements include better prediction accuracy and increases in input size that do not lower model performance.

Text-MGNN: Recent advancements in using GNNs for text classification have shown promising improvements in handling complex textual data. The study by (Gu et al, 2023) introduces the Text-MGNN model, which enhances text classification by constructing Multi-Granular Topic-Aware (MGTA) graphs to mitigate the negative effects of polysemy words and improve class-aware representation learning through the integration of word, document, and topic nodes. Another significant contribution is the work by (Huang et. al, 2024) on Residual Graph Attention Networks (ResGAT), which incorporates residual connections into GATs to enhance the flow of information and address the vanishing gradient problem, thereby improving classification accuracy in academic paper datasets.

Methodology

Using natural language processing (NLP) techniques, Pytorch Geometric's implementation of GAT and the GATv2 layer, and books available in the public domain, experiments were performed to ascertain if the structure of books could be used to identify their genre and if the size of graph inputs can affect model performance.

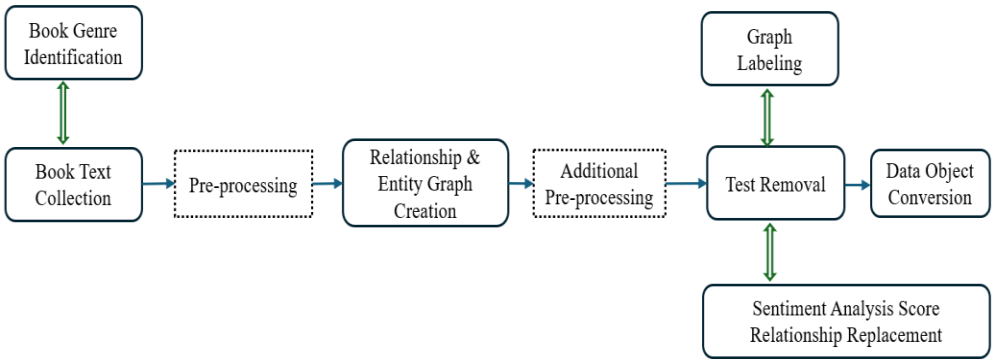


Figure 2: A diagram showing the complete project workflow with simultaneous steps denoted with double parallel lines.

Graph Construction: Graph construction underwent many iterations, in its current form, entities and their relationships are defined within book text and then further preprocessed before being converted from Networkx graphs into a Pytorch Geometric Data object.

1. **Sourcing Book Data** : 132 Books were downloaded from Project Gutenberg. Project Gutenberg is a website that hosts books with expired copyright. Using their search functions a variety of books were selected. Selected books were downloaded via plaintext text files.

2. **Determining Genre.** 12 Book genres were selected (poetry, mystery, horror, philosophy, science, science fiction, history, historical fiction, children's, adventure, fantasy, romance). Books were assigned a given genre based on the top user purported genre on the corresponding Goodreads.com page. These text labels were later encoded using sklearn's label encoder.

3. **Identifying Entities and Relationships**. Two different methods were used to identify entities and relationships. The first method followed the knowledge graph creation algorithm created by Nagesh Singh Chauhan, 2021. This algorithm pattern matches sentence structure to identify entities and relationships. The second method utilizes Natural Language Toolkit or NLTK to recognize and extract named entities (Bird,S., et al., 2019). This implementation used for graph generation also included a custom version of the semi_rel2reldict function by Liling Tan a.k.a alvas, 2016, former lecturer and current Senior Machine Learning Research Engineer at Apple (Tan, 2018).

4. **Graph Creation:** The graphs created using the aforementioned methods are pictured in figure 3, and statistically described in the Dataset Statistics section. In addition to the knowledge graph and named entity methods, one dataset also utilized a graph pruning algorithm implemented after the three preceding steps. Described in pseudo code 1, the algorithm pruned nodes based on their degree.

5. **Text Removal:** For the GAT model to process the graph data, all text had to be eliminated. During this process, encoded labels were added to their corresponding graph, entity text was removed in favor of chronological number replacement, and relationship text strings were replaced by sentiment analysis scores generated by NLTK's VADER (Valence Aware Dictionary and sEntiment Reasoner) (see pseudo code 2). VADER uses its Sentiment Intensity Analyzer (SIA) to score a given string of text's sentiment, negative, neutral, positive or compound; with compound representing the intensity of positive or negative sentiment. These four values are put into four

columns that represent four edge features. Finally, node features are necessary so node degree is used and added in during this step.

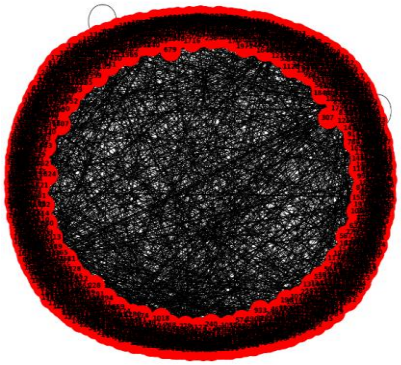
Graph Construction Algorithms: The graphs created using the aforementioned methods are pictured in figure 3 in Appendix, and statistically described in the Dataset Statistics section. In addition to the knowledge graph and named entity methods, one dataset also utilized a graph pruning algorithm. Seen below, the algorithm pruned nodes based on their degree.

```
PrunedGraph(G)
    For each graph in dataset G
        Remove = TRUE
        While Remove == TRUE
            For each node in graph
                If node degree < average
                        Add node to Removal List
                If Removal List is not empty
                            Remove Nodes
                            Remove = True
                Else
                        Remove = False
```

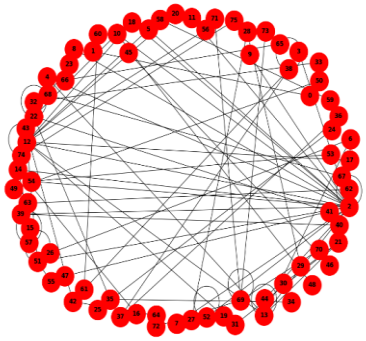
Pseudo code 1: Pruned Graph

```
TextRemoval(G)
    Updated Graphs = [ ]
    For each graph in dataset G
        New Edges = Relationship Sentiment Score
        Remove Current Edge Feature
        Graph Edge Features = New Edges
        Convert Node Labels to Integer
        For each node in graph
                Node Feature = Node Degree
        Add graph to Updated Graphs
    Return Updated Graphs
```

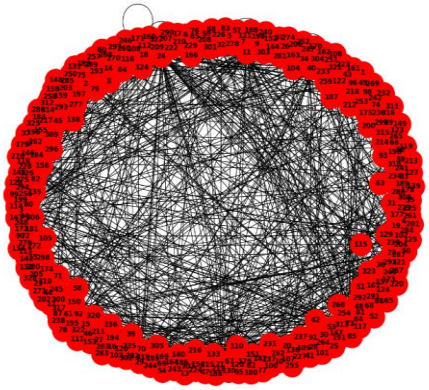
Pseudo code 2: Text Removal



(a)Full book post text removal



(b) Named Entities post text removal



(c) Pruned Graph Post Text Removal

Figure 4. Example of Constructed Graphs post text removal.

As seen in figure 4, the graphs do not change outward shape. However, as aforementioned, their internal features do change. Table 1 displays a selection of nodes and edge features for the dataset created using named entity extraction. Here it is shown that post text removal there are now four edge features, as described above holding numerical sentiment scores and sequential numbers representing differing entities.

Table 1: A selection of node and edge features viewed before and after the text removal step

	Before Text Removal	After Text Removal
Node	'odysseus', 'atlas', 'ulysses', 'book_xiii', 'odyssey', 'authoress', 'butler', 'mercury', 'minerva', 'laertes', 'telemachus', 'jove', 'amphitrite', 'patroclus', 'antilochus', 'pisistratus', 'echephron', 'aretus', 'perseus', 'book', 'menelaus', 'troy', 'helen', 'anticlus', 'aegisthus', 'proteus', 'penelope', 'ceres', 'iasion', 'nausicaa', 'queen_arete', 'echeneus', 'alcinous', 'demodocus'	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75
Edge Features	'odysseus', 'atlas', {'edge': ', the hapless one , who far from his friends this long while suffereth affliction in a sea-girt isle , where is the navel of the sea , a woodland isle , and therein a goddess hath her habitation , the daughter of the wizard'}), ('ulysses', 'book_xiii', {'edge': 'wakes in the middle of line 187 ,'})	(0, 1, {'neg_rel': 0.061, 'neu_rel': 0.861, 'pos_rel': 0.078, 'compound_rel': 0.1779}), (2, 3, {'neg_rel': 0.406, 'neu_rel': 0.594, 'pos_rel': 0.0, 'compound_rel': -0.6249}), (2, 4, {'neg_rel': 0.127, 'neu_rel': 0.767, 'pos_rel': 0.106, 'compound_rel': -0.1779}), (2, 5, {'neg_rel': 0.212, 'neu_rel': 0.726, 'pos_rel': 0.061, 'compound_rel': -0.7964})

Pruning Algorithm : The pruning algorithm (seen in pseudo code 1), while simple in implementation, provides great insights into the book data. In particular, using the knowledge graph approach on the full data, it allows for the isolation of key relationships and entities. Behaving as an additional data processing step for a dataset created using the knowledge graph creation algorithm on the full text. The idea being that major entities would have many connections, and minor entities would have few, and that relationships between major entities would carry more weight in regard to prediction accuracy as opposed to relationships involving minor entities. Similar logic can be seen when reading a book's summary.

Summaries often carry only major details involving main plot points, main characters, and important events. Compared to the other datasets, the PrunedGutenGraph dataset takes into account the entire text, only constrained by the amount of relationships a given entity is a part of. Other datasets only look at named entities or do not use the full text. The pruning algorithm, in effect, allows for the targeting of pertinent data and removal of extraneous data in a way that lowers the impact that a large text graph has on the model, while not sacrificing accuracy.

Converting to Pytorch Datasets : Following the removal of texts from the graphs, the Networkx graphs were then converted into Pytorch Data objects (see pseudo code 3). Pytorch Data objects use tensors so conversions were mostly straight forward and primarily included just changing the datatype of the various attributes. The only exception to this was the edge_index attribute which required constructing and transposing a list of edge nodes. After their creation, lists of datasets were converted into Pytorch datasets using the implementation found within the documentation (PyG Team, (n.d.)).

```
PytorchConversion(G)
    Converted Graphs = [ ]
   For each graph in dataset G
        X = tensor(node attributes)
         Edge Index = tensor (Edge List)^T
         Edge Attr = tensor([List Attribute 1, List Attribute 2, List Attribute 3, List Attribute 4])
         Add Data( X,Edge Index, Edge Attr, Graph Label) to Converted Graphs
  Return Converted Graphs
```

Pseudo code 3: Converting to Pytorch Datasets

Dataset Statistics. For the experiment three different datasets were generated and then saved as pickle files. The first dataset, Mid50GutenGraphs, contains graphs based on the knowledge graph creation algorithm applied to the middle 50% amount of book text. The second dataset, BEGutenGraphs, contains graphs created using NLTK's entity recognition algorithm on the entire book text. The third dataset, PrunedGutenGraphs is created by applying a pruning algorithm (refer to pseudo code 1) to a dataset generated by applying the knowledge graph creation algorithm on the entire book text.

Table 2, shows the total number of nodes and edges for each dataset. Mid50GutenGraphs is by far the largest, with PrunedGutenGraphs coming in second. Both of these graphs come from using the knowledge graph creation algorithm. Also interesting is the discrepancy in number of edges in regards to the BEGutenGraphs dataset. BEGutenGraphs may share a similar number of total nodes with Pruned GutenGraphs, but because BEGutenGraphs requires NLTK to recognise two named entities before it identifies a relationship between them, it has a lower number of edges.

Table 2: Dataset Statistics for Node Number and Edge Number

Dataset	Total Number of Nodes	Total Number of Edges
Mid50GutenGraphs	518036	259018
BEGutenGraphs	16092	8046
PrunedGutenGraphs	19567	142121

Table 2 shows the total number of nodes and edges for each dataset. Mid50GutenGraphs is by far the largest, with PrunedGutenGraphs coming in second. Both of these graphs come from using the knowledge graph creation algorithm. Also interesting is the discrepancy in number of edges in regards to the BEGutenGraphs dataset. BEGutenGraphs may share a similar number of total nodes with Pruned GutenGraphs, but because BEGutenGraphs requires NLTK to recognise two named entities before it identifies a relationship between them, it has a lower number of edges.

Experimental Setup

Flow Diagram

Because there are only 132 graphs in each dataset 10 fold K-Fold Cross Validation was implemented (see figure 6). After each fold was finished training, test accuracy was measured. The final list of accuracies was then analyzed to measure average, range and spread. This process, illustrated in figure 5, was done repeatedly for several different model configurations.

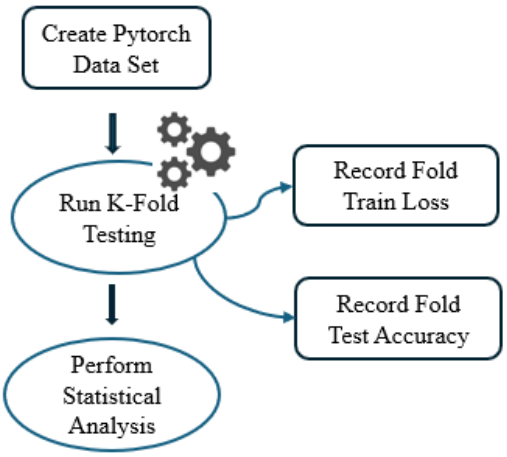


Figure 5.: A Diagram showing the simplified experimental workflow

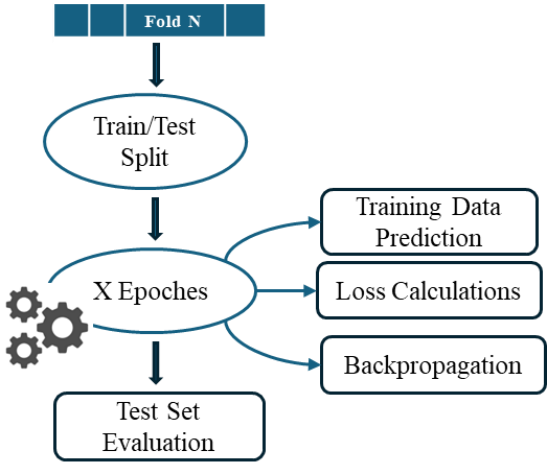


Figure 6: The flow of execution for each model run

Model Choice Considerations: Not all GNNs are appropriate for all applications. Because relational data is represented within the edge features the chosen model must utilize edge features for its prediction. The datasets were converted for use with Pytorch Geometric. Pytorch Geometric contains a lot of in depth documentation regarding various GNN models. With that in mind, the GAT model with GATv2 layers was chosen. The model was then implemented following the Pytorch Geometric GAT tutorial (Longa, A., (n.d.).) along with graph level classification changes implemented by Dr. Matthias Fey a.k.a rusty1s(Fey, M., 2021), founder of Pytorch Geometric(Fey, M., (n.d.)).

Loss Function: The model utilizes the NLL Loss function. NLL Loss or negative log likelihood loss (PyTorch Contributors, (n.d.)) is used for classification problems and takes an input of given log probabilities for each class.

$$l(x,y) = -\frac{1}{N} \cdot \sum_{n=1}^{N} \omega * y_n \, log(p_{yn})$$

x *is the input,* y *is the target,* N *is the Number of samples,*
w *is an optional weight for each sample*

Configurations: There are four different experimental model configurations. At their base, each configuration uses a batch size of 15, learning rate of 0.005, and three layers in the shape (1,13), (104,13), and (104,13). Configuration 1 uses 10 epochs. Configuration 2 uses 5 epochs. Configuration 3 uses 15 epochs. Configuration 4 uses 20 epochs.

Results and Discussion

Results Collection: As seen in figure 5, K-Fold validation was used to ascertain the preformation of a given configuration. During K-Fold Cross Validation, the data is randomly split between a training and testing set. This process is repeated across ten folds, each using a different test set. In effect K-Fold Cross

Validation allowed us to gain an understanding on how each model configuration performs on data it has not seen before, utilizing 10 different randomly generated test sets. In particular, statistical data such as mean, median, mode, IQR, standard deviation, min, and max were recorded for each fold. This collection of data allowed us to analyze spread, and overall accuracy for each fold

Table 3: Overview of Experimental Results

Configuration	Average Mean Accuracy	Max Accuracy	Min Accuracy	Max IQR	Most Accurate Dataset
1 (10 epochs)	28.993%	76.923% (PGG)	0.000% (BEGG)	45.467% (PGG)	PGG
2 (5 epochs)	21.575%	61.538% (M50GG)	0.000% (PGG & BEGG)	19.093% (M50GG)	M50GG
3 (15 epochs)	31.850%	76.923% (PGG)	0.000% (BEGG)	40.385% (PGG)	PGG
4 (20 epochs)	39.286%	100.000% (PGG)	0.000% (BEGG)	56.593% (PGG)	PGG

Table 3 shows an overall view of the results from the experiment. After Averaging the mean accuracy values for each dataset configuration run, configuration 4 has the highest accuracy, while configuration 2, which only utilizes 5 epochs, has the lowest. This is reflected again in the max accuracy and max IQR columns. In at least one-fold of every configuration, BEGutenGraphs (BEGG) returned a test accuracy of 0%. In contrast, Mid50GutenGraphs (M50GG) never had 0% test accuracy. Across all test configurations, the PrunedGutenGraphs (PGG) had the highest max accuracy, correctly predicting the genres of all the books in the applied test set, in one-fold.

Model Config 1 (10 epochs): Table 4 shows the accuracy analysis for model configuration 1. Here as seen in table 3, the PrunedGutenGraphs dataset has the highest max and average accuracy across all folds, while the Mid50GutenGraphs dataset has the second highest max and average accuracy. The BEGutenGraphs dataset falls to third in both of the previous aforementioned metrics, while also having the lowest min accuracy. Overall, the results for model configuration 1 show that, at 10 epochs, the PrunedGutenGraphs dataset, performs the best.

Table 4: Overview of Model Configuration 1 Experimental Results

Dataset	Average Accuracy	Standard Deviation	Max Accuracy	Min Accuracy	Median Accuracy	IQR
M50GG	30.440%	20.300	61.538%	7.692%	21.978%	28.846
BEGG	13.736%	10.745	30.769%	0.000%	15.385%	17.583
PGG	42.802%	24.673	76.923%	14.286%	38.462%	45.467

a. **M50GG.** Figure 7 shows the spread of accuracy data gathered for M50GG. Here it is shown that for the majority of folds the accuracy lies somewhere between 15.385% and 44.231%. This is reflected in table 4 with a given average accuracy of about 30% and an IQR of about 29. Figure 7 also shows the distance between the majority of data and the max and min fold accuracy. Here it is shown that the majority of data lies closer to the minimum than the maximum recorded accuracy. This is reflected in both the mode and the median values (15.385% and 21.978% respectively).

b. **BEGG.** Figure 8 shows the spread of accuracy data gathered for BEGG. Here it is shown that for the majority of folds the accuracy lies somewhere between 3.572% and 21.154%. This is reflected in table 4 with a given average accuracy of about 14% and an IQR of about 18. Figure 8 also shows the distance between the majority of data and the max and min fold accuracies, which again, like in figure 7, shows a closeness between the minimum recorded accuracy value and the majority of record accuracy values. This is reflected in the mode values of 0% and 15.385%, and in the median value 15.385.

c. **PGG.** Figure 9 shows the spread of accuracy data gathered for PGG. Here it is shown that for the majority of folds the accuracy lies somewhere between 21.841% and 67.308%. This is reflected in table 4, with a given accuracy of about 43% and an IQR of about 45. Unlike the BEGG and M50GG test results, the spread of data is very even and the majority of fold accuracies lie in a close to equal distance between the minimum and maximum recorded values. This is reflected in the mode (69.231%) and the median (38.462%).

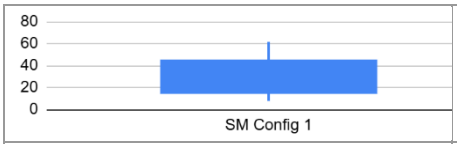


Figure 7: A candlestick graph showing the spread of the M50GG accuracy across all folds for Configuration 1.

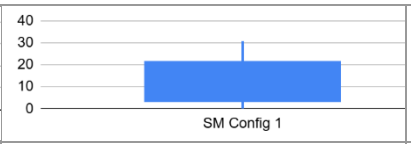


Figure 8.: A candlestick graph showing the spread of the BEGG accuracy across all folds for Configuration 1.

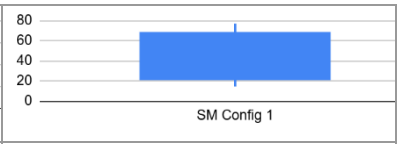


Figure 9: A candlestick graph showing the spread of the PGG accuracy across all folds for Configuration 1.

Model Config 2 (5 epochs): Table 5 shows the accuracy analysis for model configuration 1. Here as seen in table 2, the Mid50GutenGraph dataset has the highest max accuracy and the highest average accuracy. It also is the only dataset to have a min accuracy above 0.000%. The PrunedGutenGraphs dataset has the second highest average and max accuracy, with BEGutenGraphs again falling to third. Unlike in table 4 where the top two performing datasets are close in accuracy with a large gap to third, accuracy numbers are much more closer to each other across all folds for all datasets. Overall, the results for the model configuration 2 show that, at 5 epochs, the Mid50GutenGraphs dataset performs the best.

Table 5: Overview of Model Configuration 2 Experimental Results

Dataset	Average Accuracy	Standard Deviation	Max Accuracy	Min Accuracy	Median Accuracy	IQR
M50GG	27.308%	14.947	61.538%	14.286%	23.077%	19.093
BEGG	14.506%	10.623	38.462%	0.000%	15.385%	8.105
PGG	22.912%	13.085	46.154%	0.000%	23.077%	13.873

a. **M50GG.** Figure 10 illustrates that for the majority of folds the recorded accuracy lies somewhere between 15.385% and 34.478%. Here the majority of collected data is extremely close to the minimum value of 14.286%. This is reflected in table 5, with a given average accuracy of about 27% and an IQR of about 19. Figure 10 also shows many recorded accuracies lies very far from the record maximum value. This is reflected in the mode value of 15.385% and the median value 23.077%.

b. **BEGG.** Figure 11 shows that the majority of recorded fold accuracy values lie between about 7.280% and 15.385%. This is reflected in table 5, with a given average accuracy of about 15% and an IQR of about 8. Compared to the M50GG results the majority of data lies above the minimum but is still closer to the minimum value (0%) than the maximum value (38.462%). This is reflected in the mode and median value of 15.385%.

c. **PGG.** Figure 12 shows most record fold accuracies lies between 16.896% and 30.769%. This is reflected in table 5, with an average accuracy of about 23 and an IQR of about 14. Like in the previous PGG test, the results show a more equal spread between the maximum and minimum values when compared to the other datasets. This is reflected in the mode and median values of 30.769% and 23.077%, respectively.

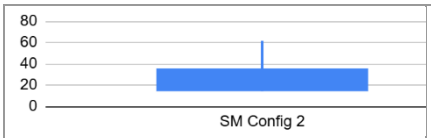


Figure 10: A candlestick graph showing the spread of M50GG accuracy data across all folds for Configuration 2

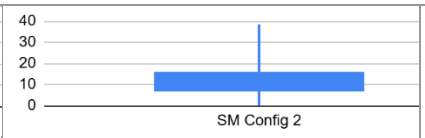


Figure 11: A candlestick graph showing the spread of BEGG accuracy data across all folds for Configuration 2.

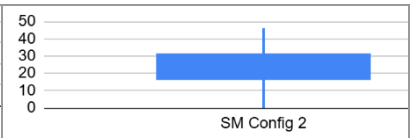


Figure 12: A candlestick graph showing the spread of PGG accuracy data across all folds for Configuration 2.

Model Config 3 (15 epochs): Table 6 shows the accuracy analysis for model configuration 3. Here as seen in table 3, the PrunedGutenGraphs dataset has the highest max and average accuracy, with Mid50GutenGraphs having the second highest values in both metrics. These results mirror the results in table 4, however, across all columns every dataset (except PrunedGutenGraphs) performs better, with higher accuracy. Overall, the results for the model configuration 3 show that, at 15 epochs, the PrunedGutenGraphs dataset performs best.

Table 6: Overview of Model Configuration 3 Experimental Results

Dataset	Average Accuracy	Standard Deviation	Max Accuracy	Min Accuracy	Median Accuracy	IQR
M50GG	32.088%	15.933	69.231%	7.143%	30.769%	11.539
BEGG	22.088%	16.685	46.154%	0.000%	21.978%	26.924
PGG	41.374%	25.899	76.923%	7.143%	30.769%	40.385

a. **M50GG.** Figure 13 shows, many fold accuracy values lies between 25% and 36.539%. This is reflected in table 6, with a given average accuracy of about 32% and an IQR of about 12. The figure also shows that many recorded values lie closer to the minimum recorder value than the maximum recorded value. This is reflected in the mode and median value of 30.769%.

b. **BEGG.** Figure 14 shows many fold accuracy values lie between 9.615% and 36.539%. This is reflected in table 6, with a given accuracy of about 22% and an IQR of about 27. The Figure also shows many recorded values lie in a close to equal distance between the minimum and maximum values, in contrast to the configuration 1 and 2 BEGG test results. This even spread is reflected in the mode values of 0%, 15.385%, and 38.462% and the median value of 21.978%.

c. **PGG.** Figure 15 illustrates that the vast majority of fold accuracy values lie between 25% and 65.385%. This is reflected in table 6, with a given accuracy of about 41% and an IQR of about 40. The figure also shows that the majority of data is slightly closer to the maximum value instead of the minimum value. This is reflected in the mode and median value of 30.769%.

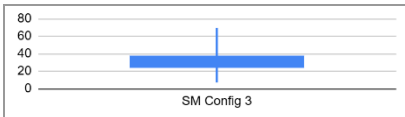


Figure 13: A candlestick graph showing the spread of M50GG accuracy data across all folds for Configuration 3.

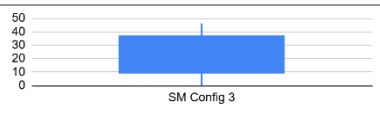


Figure 14: A candlestick graph showing the spread of BEGG accuracy data across all folds for Configuration 3.

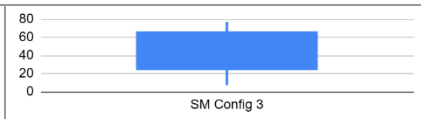


Figure 15: A candlestick graph showing the spread of PGG accuracy data across all folds for Configuration 3.

Even though both PGG and M50GG test results show the same median and mode value, the difference in spread is due to the concentration of data. In M50GG tests for configuration 3, the data is concentrated around the median value with an IQR of only 11.539. In comparison, the IQR for the PGG spread is 40.385, almost 4 times a wider range between the first and third quartiles.

Model Config 4 (20 epochs): Table 7 shows the accuracy analysis for model configuration 4. Here as seen in table 3, the PrunedGutenGraphs dataset has the highest max and average accuracy, with Mid50GutenGraphs having the second highest values in both metrics. These results show a dramatic increase in accuracy compared to the other tests. Overall, the results for the model configuration 4 show that, at 20 epochs, the PrunedGutenGraphs dataset performs best.

Table 7: Overview of Model Configuration 3 Experimental Results

Dataset	Average Accuracy	Standard Deviation	Max Accuracy	Min Accuracy	Median Accuracy	IQR
M50GG	34.341%	19.419	61.538%	7.143%	33.517%	28.846
BEGG	27.583%	23.623	69.231%	0.000%	19.231%	29.122
PGG	55.934%	33.546	100.000%	7.692%	61.538%	56.593

a. **M50GG.** Figure 16 shows that the majority of fold accuracy values lie between 17.308% and 46.154%. This is reflected in table 7, with an average accuracy of about 34% and an IQR of about 29. Figure 16 also shows that the majority of data sits a close to equal distance between the minimum and maximum value. This is reflected in the mode values of 15.385%, 46.154%, and 61.538% and the median value of 33.517%.

b. **BEGG.** Figure 17 shows that the majority of recorded fold accuracy values lie between 9.341% and 38.462%. This is reflected in table 7, with an average accuracy of about 28% and an IQR of about 29. The figure also shows that the majority of accuracy values lie closer to the minimum value than the maximum value. This is reflected in the mode values of 7.692% and 38.462%, and the median value of 19.231%.

c. **PGG.** Figure 18 shows the majority of recorded fold accuracy values lie between 24.176% and 80.769%. This is reflected in table 7, with an average accuracy of about 56% and an IQR of about 57. The figure also shows that the majority of data sits equally between the maximum and minimum values. This is reflected in the mode value of 14.286% and the mode and median value of 61.538%.

Figure 16: A candlestick graph showing the spread of M50GG accuracy data across all folds for Configuration 4.

Figure 17: A candlestick graph showing the spread of BEGG accuracy data across all folds for Configuration 4.

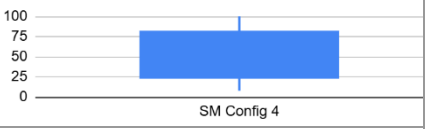


Figure 18: A candlestick graph showing the spread of PGG accuracy data across all folds for Configuration 4.

Limitations

Though care was taken to collect accurate and meaningful data, because of the size of the text, we were unable to run tests on graphs generated based on the entirety (non-reduced) book text. Large epoch testing (100+) was also unable to be completed due to graph size and complexity of the Mid50GutenGraph and previously mentioned full book text datasets. Besides the changes in dataset the only changed variable within the experiment was the number of epochs. Though this had a large effect on recorded accuracy, it is possible that changing other model parameters, like batchsize, learning rate, model layers, and even the model itself, could lead to different results and conclusions. Furthermore, changes in graph construction could also have an effect . Finally there are many different genres and further testing is needed to prove if this method of genre prediction works across all cases.

Conclusions

In this study, we converted book text, gathered from Project Gutenberg, into graphs using various algorithms. One algorithm (seen in pseudo code 1) used a pruning approach to isolate important relationships found between entities in the book text. Other algorithms studied utilized only named entities or relationships present in the middle 50% of the book text. The converted book graph data was then used with a Graph Attention Network (GAT) model to achieve the classification of book genres. Testing shows that while Mid50GutenGraphs and PrunedGutenGraphs remained competitive across 10 and 15 epochs, when the model only underwent 5, the dataset with the larger graph size (more entities and relationships) had much better performance. Tests also showed that an increase in the number of epochs corresponded with an increase in accuracy across all sizes of graphs, with the 20 epoch configuration performing the best.

Overall, the success of the experiment illustrates that the entities and relationships present in books can be used to predict a book's genre and that text-based graph construction methods can be scaled up to work with full book text. However, this scalability when applied to GNN graph inputs shows that changing the size of graphs changes the information held within them and thus affects accuracy. In this study, the model configuration with 20 epochs, using the PrunedGutenGraphs dataset, performed the best, peaking at

100% accuracy, averaging out to almost 56% accuracy across all folds. This indicated that the model can perform exceptionally well on unseen data using that configuration. From this we conclude that pruning constructed book graphs is better than other book graph construction methods including isolating named entities and using only the middle 50% of the text. Furthermore, our research provides a strong example of how GATs can be utilized for the classification of complex data sets.

For future work, analyzing the correlation between graph construction and GAT accuracy promises to enhance overall accuracy. Identifying cases of overfitting and underfitting, coupled with proposing effective hyperparameter tuning methods, will further optimize the model's performance.

References

Bhakta, S.S. (2024, January 6). Recursive Neural Network in Deep Learning. GeeksforGeeks. https://www.geeksforgeeks.org/recursive-neural-network-in-deep-learning/

Bird,S., Klein, E., & Loper, E. (2019) Natural Language Processing with Python. https://www.nltk.org/book/ch07.html#ref-chunkex-sent

Brody, S., Alon, U., Yahav, E.:How Attentive are Graph Attention Networks. ICLR (2022).

Clabaugh, C., Myszewski, D., & Pang, J. (n.d.). Neural Networks - History. Neural Networks - Sophomore College 2000. Retrieved June 10, 2024 from https://cs.stanford.edu/people/eroberts/courses/soco/projects/neural-networks/History/history1.html

Chauhan, N.S. (2021, February 11). Build knowledge graph using python. Kaggle. https://www.kaggle.com/code/nageshsingh/build-knowledge-graph-using-python

Fey, M. (2021, November 17). GAT for graph classification#3516. Github. https://github.com/pyg-team/pytorch_geometric/discussions/3516#discussioncomment-1656045

Fey, M. (n.d.). Matthias Fey. Github. Retrieved June 11, 2024 from https://github.com/rusty1s

GeeksforGeeks. (2024, March 14). Introduction to Convolution Neural Network.https://www.geeksforgeeks.org/introduction-convolution-neural-network/

Ghorakavi, V. (2024, January 3). What is a neural network?. GeeksforGeeks. ,https://www.geeksforgeeks.org/neural-networks-a-beginners-guide/

Gilmer, J., Schoenholz, S. S., Riley, S.F., Vinyals, O., & Dahl, G. E. (2017). Neural Message Passing for Quantum Chemistry. InInternational conference on machine learning, 1263-1272.

Gu, Y., Wang, Y., Zhang, H. R., Wu, J., & Gu, X. (2023). Enhancing text classification by graph neural networks with multi-granular topic-aware graph. IEEE Access, 11, 20169-20183

Hashemi-Pour, C.,Lutkevich, B. (2024). What is Bert?. TechTarget. https://www.techtarget.com/searchenterpriseai/definition/BERT-language-model

Huang, L. , Ma, D., Zhang, X.., & Wang, H. (2019). Text Level Graph Neural Network for Text Classification. arXiv preprint arXiv:1910.02356.

Huang, X., Wu, Z., Wang, G., Li, Z., Luo, Y., & Wu, X. (2024). ResGAT: an improved graph neural network based on multi-head attention mechanism and residual network for paper classification. Scientometrics, 1-22

Kipf, T. N., & Welling, M. (2017). Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907(https://arxiv.org/abs/1609.02907).

Lin, Y., Meng, Y., Sun, X., Han,Q., Kuang, K., Li, J., & Wu, F. (2021). BertGCN: Transductive Text Classification by Combining GCN and Bert. arXiv preprint arXiv:2105.05727

Longa, A. (n.d.). Tutorial3. colab. Retrieved June 11, 2024 from https://colab.research.google.com/github/AntonioLonga/PytorchGeometricTutorial/blob/main/Tutorial3/Tutorial3.ipynb

PyG Team. (n.d.). Creating Graph Datasets. Retrieved June 10, 2024 from, https://pytorch-geometric.readthedocs.io/en/latest/tutorial/create_dataset.html

PyTorch Contributors. (n.d.). NLLLoss. Retrieved June 18, 2024, from https://pytorch.org/docs/stable/generated/torch.nn.NLLLoss.html

Sanchez-Lengeling, B., Reif, E., Pearce, A., & Wiltschko, A.B. (2021, September 2). A Gentle Introduction to Graph Neural Networks. distill. https://distill.pub/2021/gnn-intro/

Shen, Y., Cheng, X., Li, H., & Liu, J. (2020). Connecting the dots: What graph-based text representations work best for text classification using graph neural networks? arXiv preprint arXiv:2305.14578(https://arxiv.org/abs/2210.05999)

Tan, L. (2016, November 9). NLTK relation extraction returns nothing. Stack Overflow. https://stackoverflow.com/a/40498962

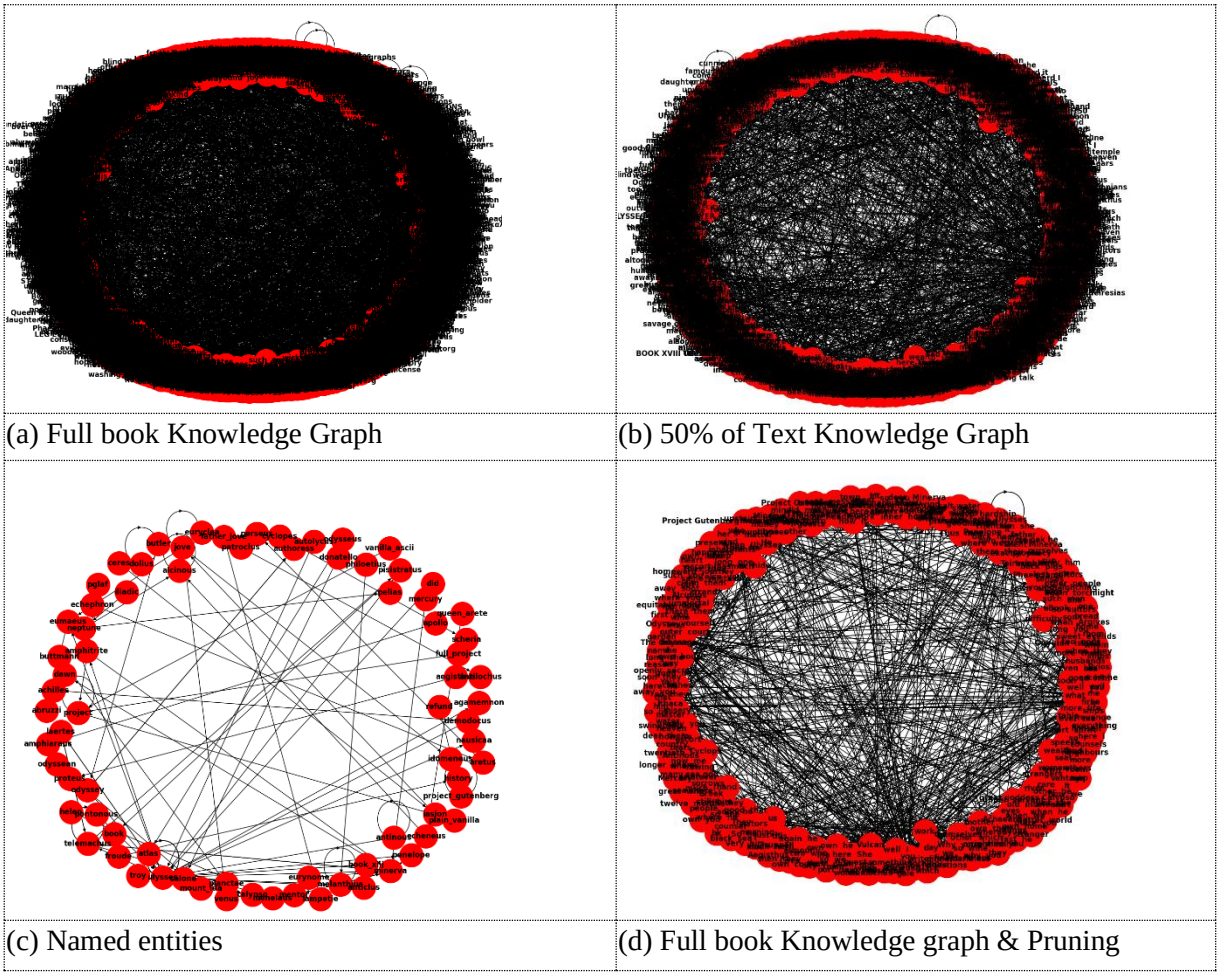
Tan, L. (n.d.). Liling Tan. Alvations. Retrieved June 11, 2024 from https://alvations.bitbucket.io/

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., & Bengio, Y. (2018). Graph attention networks. arXiv preprint arXiv:1710.10903.

Yao, L., Mao, C., & Luo, Y. (2019). Graph Convolutional Networks for Text Classification. InProceedings of the AAAI conference on artificial intelligence 33 (1). 7370-7377

Zhou, J., Cui, G., Hu,S.,Zhang,Z.,Yang,C.,Liu,Z.,Wang,L.,Li,C., & Sun,M.. (2020). Graph neural networks:A review of methods and applications. AI Open, 1, 57-81.

Appendix



(a) Full book Knowledge Graph | (b) 50% of Text Knowledge Graph

(c) Named entities | (d) Full book Knowledge graph & Pruning

Fig 3. Various graphs of The Odyssey by Homer (a) : The graph of the full book created using the knowledge graph creation algorithm. (b): The graph of the middle 50% of text using the knowledge graph creation algorithm. (c): A graph created using identified named entities. (d): A graph created using the full text, the knowledge graph creation algorithm, and the pruning algorithm (see pseudo code 1).