Canadian Pension Plan Investment Board (CPPIB): Exploring Deep Learning Methods on the Analyst Coverage Network

Jie Yang
Department of Computer Science,
University of Toronto
alisayang@cs.toronto.edu

January 6, 2024

Industry Supervisor:
Yixing Zhang,
Investment Science,
Canadian Pension Plan Investment Board
yzhang@cppib.com

Academic Supervisor:
Christina C. Christara,
Department of Computer Science,
University of Toronto
ccc@cs.toronto.edu

This report is presented to partially fulfill the degree requirements for the Masters of Science in Applied Computing, with expected degree completion in January 2024

Abstract

Graph datasets can be a rich source of information in fields such as Finance, since they can capture relationships that may not be evident in traditional tabular datasets. The Analyst Coverage Network (ACN) is a graph dataset where each node corresponds to a company, and edges between two nodes/companies represent the number of equity analysts covering both companies. We believe the ACN contains useful information on company similarity because analysts possess domain expertise and therefore are more likely to cover companies that are related in some dimension. Being able to group similar companies is an important task in Finance, as they often exhibit correlated returns. Graph Neural Networks (GNNs) are popular models used to extract information from graphs and both supervised (Graph Convolutional Network) and unsupervised (node2vec) techniques have demonstrated strong performance for many deep learning tasks. Applying both techniques to the ACN dataset gives us a broader understanding of their individual strengths and weaknesses. We demonstrate that GNNs can effectively extract meaningful features from the ACN graph. Specifically, we observe that the node2vec algorithm, which is an unsupervised technique, is simpler to train and produces embeddings that encode similar information to the Global Industry Classification Standard (GICS). These embeddings can then be used for a variety of downstream forecasting tasks. In contrast, the supervised technique is target-specific and can generate predictions directly, but it is harder to train and does not perform well with limited number of input features.

Contents

1 Introduction

1.1 Background Information

Graph datasets can be a rich source of information but they are hard to process and can take up a lot of space (e.g. traditional adjacency matrices have a space complexity of $\mathcal{O}(N^2)$ where N is the number of vertices in a graph). They also tend to be sparse and cannot be used in downstream forecasting tasks directly. Deep learning methods have become very popular in the last few years, since they seem to be capable of modeling complex real work interactions. They have been used in a variety of applications from text citation to protein interactions [21] and even stock prediction in Finance.

The Canadian Pension Plan Investment Board is a global investment management organization tasked with investing the funds of the Canadian Pension Plan (CPP) to maintain its long-term sustainability. It mandate is to maximizing returns for the CPP's contributors and beneficiaries without undue risk. Exploring new techniques and datasets is important for companies like CPPIB (Canadian Pension Plan Investment Board), that manages $ 575 billion [15] in net assets, because it can bring novel and valuable insight to help make better and more informed investment decisions.

The graph dataset we will use is the Analyst Coverage Network (ACN) where each node is a company, and edges between nodes/companies represent the number of equity analysts covering both companies (e.g. there will be an edge between Google and Meta if there is a sell-side analyst writing a report on both companies).The assumption is that companies that are covered by the same equity analysts will exhibit more similarities than companies that are not covered by the same analysts. This is because analysts tend to have domain expertise, e.g. a technology analyst will probably not cover a mining company. This can offer a more nuance understanding of a company's relationship to another compared to a more traditional classification that tends to remain static through time.

1.2 Research Goals and Outcomes

In this paper, we will review a few deep learning methods for extracting information from graphs. We can separate these methods into two categories which are the unsupervised and the supervised methods. The main difference between the two is whether there is a target/label in the training process. An unsupervised approach will not have a label, and the goal is to faithfully encode the graph in a much lower dimensional space, while the supervised approach teaches the model to extract information from the graph to predict a specific target. For the unsupervised approach we will use the node2vec algorithm [4] and for the supervised algorithm we will use the Graph Convolutional Neural Network (GCN) [5].In this paper we will explore these two methods for predicting stock returns and downstream forecasting tasks.

Previous internal research has shown that a node2vec model with well tuned hyper-parameters can create meaningful representation of an ACN graph. The research finds that graph embeddings encapsulate similar information to the Global Industry Classification Standard (GICS) [12]. GICS classifies companies by different levels of granularity of sector, industry and sub-industry. We will extend this research to understand whether these node2vec embeddings have explanatory power on top of the GICS classification. We will also explore supervised GNN methods which will be trained/tuned for the specific task of predicting stock returns. Finally, we want to compare these output to the baseline with GICS classification. We find that both GCN and the node2vec models can create features that are additive on top of the GICS feature, but node2vec features are more expressive than the features created by the GCN. On the other hand, GCN performs poorly in the absence of a rich features library and cannot beat the baseline of just taking the weighted average.

2 Related Works

2.1 GNN in Finance

There has been a lot of work done with GNNs in Finance [19] since a lot of data in the field can be modeled with graphs. Some examples of this are knowledge graph depicting things like supplier, customer, partner and shareholder relations [9], as well as similarity based graphs such as stock correlation [7] or news co-occurrence graph [8]. There are different tasks that the GNNs can do, including node or edge level prediction. For example, in this report we are doing a node level prediction, since each node is a company and we are trying to predict the return, which is a characteristic of the node. An edge level prediction involves predicting a characteristic of an edge, for example, it could predict if two companies will merge in the next year. Depending on the task, some popular metrics for regression are Mean Square Error (MSE), Mean Absolute Error (MAE) [3] as well as other metrics more specific to the finance field, like return ratio and sharpe ratio [9]. For classification metrics we have accuracy, recall, precision, F1 and AUC which can be used for tasks like credit default prediction [6]. Research in GNN also covers a variety of models such as Graph Convolutional Networks (GCN), Graph Attention Networks (GAT), as well as incorporating other temporal models such as Recurrent Neural Networks (RNNs). Some other authors focus on creating embeddings from graphs that are later used in a downstream forecasting task, such as [16], where they used Large Language Models (LLMs) to create embeddings of each company. To the best of our knowledge, this is the first time this type of analysis has been conducted on the Analyst Coverage Network.

2.2 Node2Vec

Node2Vec borrows ideas from the Word2Vec algorithm [10] but instead of words, we have a sequence of random walks traversing the graph. The underlying idea is that, if we sample enough times from the graph, we can learn the transition matrix of the graph, and therefore encode its structure [4]. Node2Vec is an unsupervised approach therefore it optimizes the following objective function:

$$\max_f \sum_{n \in V} \log Pr(N_S(n)|f(n))$$

The function maximizes the log-probability of seeing the neighbourhood $N_S(n)$ for a company with index n conditional on the feature representation f. In practice, this can be achieved by creating multiples samples of biased random walks which are used to create a corpus. The corpus is then used to train a word2vec model based on the skip-gram architecture [10]. In the trained model, we create a vector of size d for each node. This effectively reduces the graph from size $n \times n$ to $n \times d$. Nodes that appear together in the graph will also be closer in the embedding vector space.

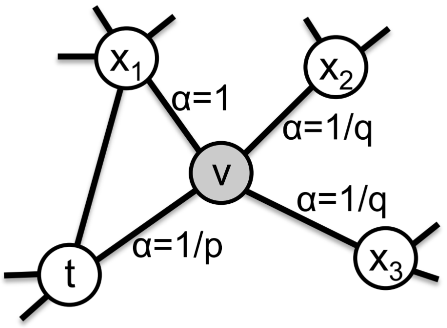


Figure 1: This graph depicts the bias random walk procedure from [4]. The previous node is t, current node is v and the next nodes can be x_1, x_2, x_3. The number p represents the return parameter and q represents the in-out parameter which make up α, the search bias, which determines the next node in the random walk.

The way we sample for the random walks is important; for example by doing a Breadth-first Sampling (BFS), we remain in the neighbourhood of the initial node n_1 while a Depth-First Sampling (DFS), involves sampling neighbours that are further out from the initial node. The node2vec algorithm is not a purely BFS or DFS sampling, rather, it has two parameters p and q, namely the return and the in-out parameter respectively, which together control how much exploration the model does. The p parameter is associated with the probability of returning to a node just visited. A high value of p reduces the probability of going back to a node that was just visited, while a high value of q implies low probability of exploring outside the node's neighbourhood. Lower values of q will encourage the model to explore more and reflects DFS behaviour, while higher values of q will mimic BFS. These two parameters p and q are hyperparmeters that need to be tuned. Some advantages of the node2vec algorithm is that we do not need labels and the embeddings created are not target specific, which means they can be used for many downstream forecasting tasks.

Previous work has been done on the Analyst Coverage Network internally at the company. The previous work finds that the node2vec algorithm is an efficient way to encode the graph into d sized embedding capable of doing well at graph reconstruction and link prediction tasks. We will use the hyperparameters from this previous study to train our node2vec model. The previous work was done for a few randomly selected dates from the whole history. This means that the results might not extrapolate to a longer history. In this report, we will extend this analysis to the whole history from 2007 to 2022.

2.3 Graph Convolutional Network (GCN)

GCNs were first introduced in 2016 by Thomas N. Kipf and Max Welling [5]. The GCN technique is based on the idea of message propagation which involves aggregating the "messages" of the neighbours. For each node, it will aggregate the features of its neighbours to create predictions. The GCN architecture takes the structure

$$H^l = \sigma(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{l-1}W^{l-1}),$$

where A is a symmetric weighted adjacency matrix and n is the number of nodes in the graph. Furthermore, $\tilde{A}$ is A with the addition of self-loops, which translates to adding 1's on the diagonal of the weighed adjacency matrix, $\tilde{D}$ is the diagonal degree matrix of $\tilde{A}$, where $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$, σ is the activation function (e.g. ReLu, ELU, etc.) and $H^{l-1}$is the output of the previous layer. When $l = 1$, meaning we are at the first layer, H^0 will be the initial features X. In general, $W^{l-1}$is the shared and learnable weight matrix where the parameters get updated by gradient descent.

The loss function we are trying to minimize is the MAE, where n is number of observations. We make one observation per company and per time unit thus the number of observations is the number of companies:

$$MAE = \frac{1}{n}\sum |y - \tilde{y}|.$$

2.4 Graph Attention Network (GAT)

Another model we will consider is the Graph Attention Network (GAT) [18] based on the attention mechanism introduced in "Attention is All You Need" [17] which is one of the most pivotal papers in the Deep Learning field. The main difference between GAT and GCN lies in the weights. In GCN, weights are introduced as input to the model while in GAT, they are learnable parameters. GAT follows the architecture

$$h_i^l = \sigma(\frac{1}{K}\sum_{k=1}^{K}\sum_{j\in N_S(i)} \alpha_{ij}^k \mathbf{W}^k h_j^{l-1}),$$

where K is number of attention heads. Each attention head learns different relationships in the data, since they are independent of each other and can be calculated in parallel. The attention heads serve to increase the number of parameters and capture complex relationships. The number of attention heads is a tunable hyperparemter. $N_S(i)$ is the neighbourhood of node with index i, which is determined by the unweighted adjacency matrix, α_{ij}^k is the normalized attention scores between i and its neighbour j, which are effectively learnable edge weights. They are then multiplied with the feature of neighbour node j and a shared and learnable weight matrix $\mathbf{W}$ that applies a linear transformation to the node features h.

3 Methodology

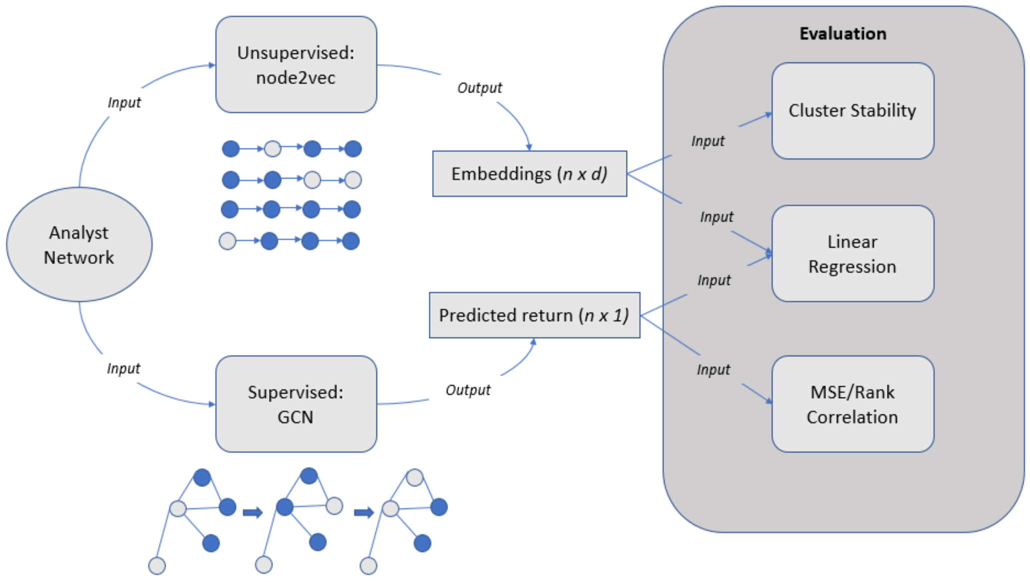


Figure 2: Methodology for evaluating the Analyst Coverage Network

3.1 Analyst Network

The Analyst Coverage Network is an un-directed weighted graph $G(V, E)$ where vertices represent companies and the presence of an edge denotes that an equity analyst is covering both companies. The weight on each edge is the number of analysts covering the company pair. The minimum weight an edge can have is 1 but the theoretical maximum is not capped, so it could be infinite but in our data set, the maximum is 31. This maximum belongs to the edge between Google and Yahoo in early 2007. The median is 1 and the average is 1.6. The number of companies n covers only US companies. We will conduct this study for the year 2007 to 2022 which will cover multiple regimes both bearish and bullish markets. The information is updated monthly.

3.2 Model Training

For the node2vec model we will create company embeddings for each data point. We will use the same hyper-parameters as the ones in the internal company paper. With the trained model we will create embeddings for each node. The node2vec algorithm takes in as input a $n \times n$ weighted adjacency matrix.

For the GCN portion we will use the GCN implementation from [1] and do hyper-parameter tuning on the first few dates of history. We select the best parameters and run with the same parameters for the rest of the history. In an ideal scenario, we would do hyperparemeter tuning with the validation set before each model, but due to time and resource constraints, we will use the former method.

The GCN can normally take in a lot of features as input, but for the fairness of the test, the only features we will use is the lagged version of our target which is the lagged 90 days cumulative return. This is more disadvantageous for the GCN, since it normally can take many features as input, while the node2vec algorithm by default does not take in any feature as input. For target, we will predict the cumulative return for the next 90 days. In this case the inputs will be features X with a size of $n \times 1$, a weighted adjacency matrix with a $n \times n$ dimension and a target Y of dimension $n \times 1$.

We will train with expanding window, with minimum 5 years of training. We will leave a buffer time between the train and the test of 120 days, which prevents look ahead bias. The validation set will be the next month out, since its used only for early stopping, and this ensures that the model train and test set are not too far apart.

3.3 Evaluation

3.3.1 Node2Vec - Cluster Stability

We will create clusters using Agglomerative Clustering [11] and Gaussian Mixture Models (GMM) [13] to evaluate the cluster stability of the embedding. We will measure the stability through time using the Adjusted Rand Index (ARI) [2]. The Rand Index (RI) measures the similarity between data clustering methods. The Adjusted Rand Index is a version of the (RI), which applies a correction for the agreement/disagreement between clusters that are due to chance. ARI ranges from 1 to -1, where 1 is complete agreement and -1 complete disagreement and 0 is a random clustering. We expect that if the embeddings are meaningful, they should be pretty stable through time. We can also treat the GICS Sector as a type of cluster and compare them to the clusters created from the embedding. We expect them to be similar, since the underlying assumptions is that equity analyst from the Analyst Coverage Network have domain expertise, and will cover companies that are in similar sectors/industries. The added benefit of the graph is that it gets updated every month while the GICS are static over time.

3.3.2 GCN - MSE and Rank Correlation

For the GCN portion we will use the weighted average of the neighbours as a baseline. This is equivalent to using $D^{-1}AX$. Recall that D is the diagonal degree matrix and A is the adjacency matrix and X is the input feature, which in our case, has the shape $n \times 1$. This is because even before training, we can get some information from the graph by just performing the GCN symmetrical normalization $\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$, which is similar to taking a weighted average of the neighbours based on their weights and connections. By comparing these output (from the weighted average and symmetric normalization) to the trained GCN, we can narrow down what is actually gained from the model training (all the hidden layers and weights).

There are some design choices we can make for the GCN. The first one is the type of propagation rule. The original GCN uses a symmetric normalization, but we can also use a different propagation rule. We can take the weighted average in the form $\tilde{D}^{-1}\tilde{A}$. The second design choice we can make is regarding the weight of the self-loops. In the original GCN implementation, we add a 1 on the diagonal to signify that the company is connected to itself but this might not be ideal for a weighted graph, where edges can have different weight depending on the size and popularity of the company they are attached to. We will explore different weighting methods for the self-loops.

3.3.3 Linear Regression

For each month we fit a linear regression with the outputs from the node2vec algorithm which is $n \times d$ and the output of the GCN which is $n \times 1$. We also include features from the GICS classification which are categorical variables that indicate if the company is in a certain Sector/Industry. The Industry classification is much more granular compared to the Sector classification since it has 74 classes while the Sector only has 11 classes.

To understand the explanatory power of the embeddings and GCNs predictions, we will use a simple linear model to fit these features to the target. We will then evaluate the MSE, R-Squared, and Adjusted R-Squared, since the models we will compare have different numbers of independent variables. Returns are notoriously hard to predict, so we will also evaluate the rank correlation, which puts more importance on the order rather than the absolute value of prediction. GICS classification and the embedding clusters will be treated as categorical variables and the embedding and GCN predictions will be treated as continuous variables. The linear regression measures the explanatory power of the features generated from the node2vec and GCN models and is similar to what [16] do.

4 Results and Discussions

4.1 Node2Vec Performance

For the node2vec method we check the cluster stability measured by the adjusted rand index (ARI) [2]. To recall, the ARI is a measure for cluster similarity and can take the value of 1 to -1, where 0 would be a completely random reshuffle of the cluster memberships and 1 means no change in the cluster membership. If we apply the ARI at each point in time t and compare it to the cluster at time $t+1$, we can estimate the cluster stability. We see in Figure 3 that the clusters are relatively stable at 0.69 and 0.68 ARI score for Agglomerative clustering and GMM clustering respectively. Agglomerative

Clustering would be preferred, since it is slightly more stable over time and the standard deviation is lower too. This suggests that the embeddings are not completely random, and are stable over time.

In the second row, we can see the similarity between the GICS sector and the clusters generated by the node2vec algorithm. We treat each GICS sector as a cluster and compare them to the clusters created from the embeddings. We see that the similarity over time is 0.41 for Agglomerative clusters. This is to be expected and corroborates our assumptions that equity analysts have domain expertise, and we see that these clusters formed are somewhat similar to GICS classifications. The advantage of the Analyst Coverage Network is that it is updated monthly, meaning it will have more recent information compared to the GICS classification that is more or less stable through history (GICS has a ARI of 0.97, it is not 1 since it has been updated a couple of times through history).

4.2 GCN Performance

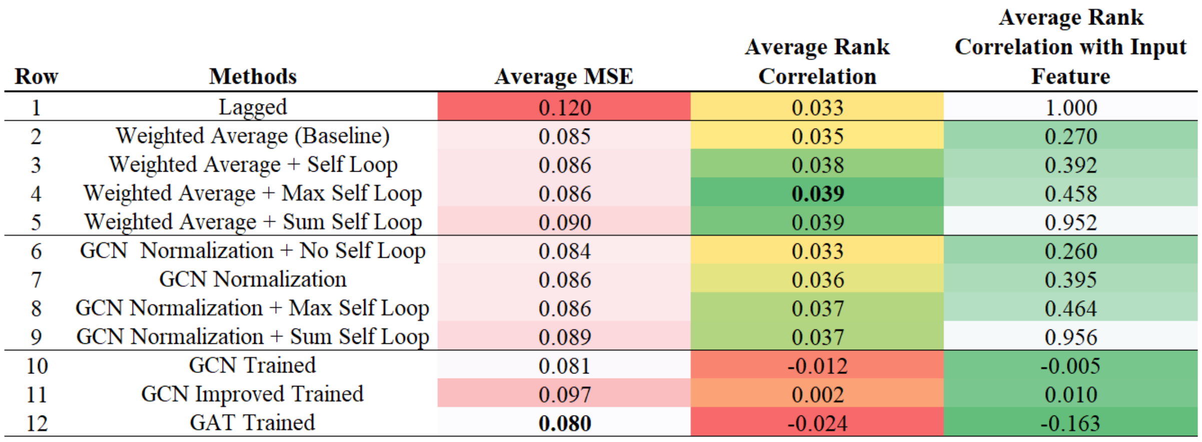
In Figure 4, going in order from top row to bottom, in the first row we see that using the lagged features as predictions gives the worse MSE, but the average Spearman correlation is positive. In the third column we have the Spearman correlation between the prediction and the input feature. Since the input feature is also the lagged feature, we have a correlation of 1.

		Agglomerative Clustering	**GMM Clustering**
ARI Over Time	Average	0.69	0.68
	Std. Dev.	0.08	0.09
ARI with GICS Sector	Average	0.41	0.46
	Std. Dev.	0.05	0.05

Figure 3: Cluster stability over time and similarity between cluster with GICS sector using Adjusted Rand Index (ARI).

In the second section of Figure 4 (rows 2-5), we have all the variations of the weighted average. We see that the applying the weighted average already lowers the MSE and increase the average Spearman correlation. Including the self-loop increases the Spearman correlation to the input feature, since its giving weights to its own features. For row 3 and 4, we are adding different modifications to the self-loop to account for the fact that our adjacency matrix is a weighted one. Instead of adding a self-loop of 1, we take the maximum/summation from the off diagonals in the weight matrix. This ensures that popular companies that have big weights on their edges are assigning enough weights to themselves.

We see that for both methods (Max and the Sum on the self-loop), increase the average Spearman correlation but also increase the MSE. The main difference between the Max and the Sum method is the average Spearman correlation to the input feature is closer to 1 for the summation method. The more weight we put on the diagonal entries of the weighed matrix the more weight we give to the input feature.



Row	Methods	Average MSE	Average Rank Correlation	Average Rank Correlation with Input Feature
1	Lagged	0.120	0.033	1.000
2	Weighted Average (Baseline)	0.085	0.035	0.270
3	Weighted Average + Self Loop	0.086	0.038	0.392
4	Weighted Average + Max Self Loop	0.086	**0.039**	0.458
5	Weighted Average + Sum Self Loop	0.090	0.039	0.952
6	GCN Normalization + No Self Loop	0.084	0.033	0.260
7	GCN Normalization	0.086	0.036	0.395
8	GCN Normalization + Max Self Loop	0.086	0.037	0.464
9	GCN Normalization + Sum Self Loop	0.089	0.037	0.956
10	GCN Trained	0.081	-0.012	-0.005
11	GCN Improved Trained	0.097	0.002	0.010
12	GAT Trained	**0.080**	-0.024	-0.163

Figure 4: Results of the GCN and GAT

When taking the summation of the off-diagonals we have a weighted adjacency matrix that is diagonally dominant, since the diagonals will always be greater than or equal to the summation of the off-diagonal matrix. Intuitively, we are saying that we always place higher weights on our own input, but this makes the output feature extremely similar to the input feature. This is an issue when creating features for downstream forecasting tasks, due to multicollinearity which can have a range of adverse effects for forecasting models [20]. For this reason, taking the maximum of the off-diagonal weights is better than taking the summation when creating feature for downstream forecasting tasks.

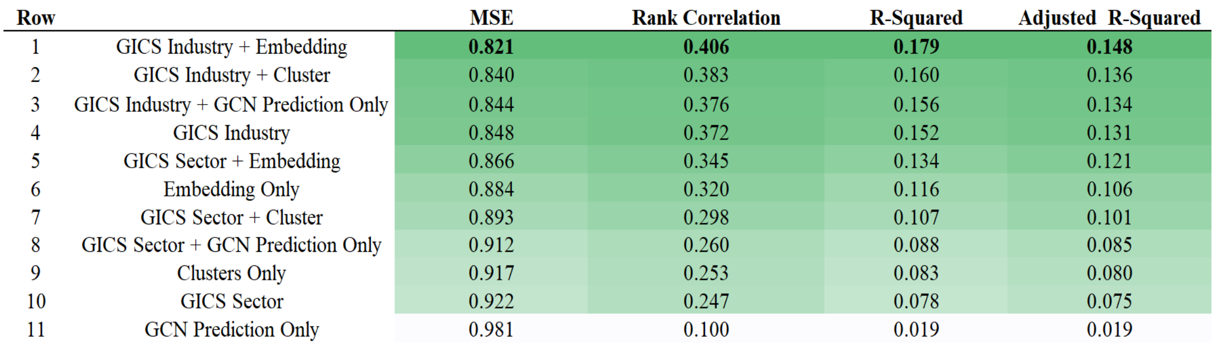
In the the third section of Figure 4, (rows 6-9) we repeat the analysis for the GCN symmetric normalization instead of the weighted average normalization. We see that it performs very similarly to the weighted average normalization. As we increase the weight on the diagonal of the weighted adjacency matrix, we also increase the weight of the input feature and this is visible from an increase in the average Spearman correlation between the prediction and the input feature. Something to notice is that the GCN normalization has similar MSE but lower Spearman correlation. This suggests, that the weighted average normalization is better than the GCN for this specific application.

For the last section of Figure 4 (rows 10-12), with the trained GCN and GAT; in row 10, the traditional GCN was trained with GCN normalization with uniform self-loop of 1's on the diagonal. We see that the MSE is reduced but the Spearman correlation is much worse than any of the previous row and is even negative. For the improved GCN, we switched out the GCN normalization for the weighted average normalization and changed the self-loop to be the max of the neighbour's weight. We see that the Spearman correlation increased, but does not beat any of the other untrained methods. The MSE is also lower. As expected the correlation with the lagged feature is higher compared to the traditional GCN. Moving onto the last row GAT achieves the lowest MSE but also has worse Spearman Correlation.

This above discussion suggests that with only lagged features as input, trained GCN/GAT does not bring more information than taking the weighted average. In fact it seems to be introducing even more noise, since the Spearman correlations are lower after training. There are multiple reasons for this poor performance. For one, returns are notoriously hard to predict and contain a lot of noise. When we train the GCN, we are adding multiple levels of complexity; as we increase the number of layers of the GCN, we are aggregating messages from high order neighbour and this can introduce noise, e.g. you might not be that similar to your friend's friend. In fact, this is a well-known problem with GCN called the over-smoothing problem[14], where predictions start to muddle as we increase the depth of the network. This might be especially prevalent with noisy data like returns.

Another reason for poor performance is the lack of features. Inside the GCN architecture we project the features onto different sized dimensions denoted by the hidden size. Since the input only has one column, expanding it to multiple hidden channels will increase noise. A strong evidence for these explanations is that when we switch to other targets and include more input features, GCN does beat the baseline of just the weighed average. We do not include these results because those features and targets are internal company information and outside the scope of the project.

4.3 Linear Regression Results



Row		MSE	Rank Correlation	R-Squared	Adjusted R-Squared
1	GICS Industry + Embedding	**0.821**	**0.406**	**0.179**	**0.148**
2	GICS Industry + Cluster	0.840	0.383	0.160	0.136
3	GICS Industry + GCN Prediction Only	0.844	0.376	0.156	0.134
4	GICS Industry	0.848	0.372	0.152	0.131
5	GICS Sector + Embedding	0.866	0.345	0.134	0.121
6	Embedding Only	0.884	0.320	0.116	0.106
7	GICS Sector + Cluster	0.893	0.298	0.107	0.101
8	GICS Sector + GCN Prediction Only	0.912	0.260	0.088	0.085
9	Clusters Only	0.917	0.253	0.083	0.080
10	GICS Sector	0.922	0.247	0.078	0.075
11	GCN Prediction Only	0.981	0.100	0.019	0.019

Figure 5: Average over time of the evaluation metrics from the linear regression metric

For each month we fit a linear regression with the output from the node2vec algorithm which is $n \times d$ and the output of the GCN which is $(n \times 1)$. 5. There are some downfalls of using a linear regression for testing the additivity of these features, but in the absence of a downstream forecasting model, linear regressions are a simple and easy way to measure the explanatory power of features.

For MSE, the lower the better while for the other metrics, the higher the better. We see that the best performing one is using the embeddings from the node2vec algorithm alongside the more granular GICS Industry classification, followed by the GICS Industry classification plus Cluster. We also see that both the features created by the GCN and the node2vec algorithm are additive on top of the GICS classification since MSE drops and the other metrics rise as we add the other features.

Using the embeddings alone seems to be more expressive than applying a clustering algorithm, since the predictive power drops when we use Cluster features.This suggests we lose some information when we cluster the embeddings. One possible reason for this is that embeddings are continuous variables, while cluster are categorical and not as expressive.

5 Conclusion and Future Research Plans

We show that GNNs can be used to extract meaningful features from the ACN graph. Both node2vec and GCN can be used to create features for downstream forecasting tasks. In the absence of a rich feature library, a node2vec algorithm is more suited for extracting meaningful features from a graph, since GCN does not perform well with only one feature. GCN also tends to have more parameters and needs longer to train. We can improve the performance of the original GCN implementation by making small changes to the weight of the self-loop (e.g. sum vs max) and the type of propagation rule (e.g. symmetric normalization vs weighted average). Directly using node2vec embeddings appears to be more expressive than creating clusters from them, as embeddings are continuous features compared to categorical nature of clusters.

Additionally, we have created and saved historical features from the ACN which can be employed for downstream forecasting tasks. The framework established to evaluate these techniques supports easy integration of other GNN models, providing a good foundation for future research. Some natural next steps for the GCN model are to include more features in the input with different time horizons (e.g. 120 days, 30 days, 1 day, etc.). For a more robust assessment of the additivity of the GNN features, it is essential to compare predictions generated by various downstream forecasting methods, both trained with and without the newly created features. Some promising models within the area of GNNs are the Temporal Graph Networks which combine different recurrent architectures and can handle the temporal aspect of the graph and returns.

References

[1] Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. In *International conference on machine learning*, pages 1725–1735. Proceedings of Machine Learning Research (PMLR), 2020.

[2] Soumita Das and Anupam Biswas. Analyzing correlation between quality and accuracy of graph clustering. *Advances in Computers*, 128:135–163, 2023.

[3] Fuli Feng, Xiangnan He, Xiang Wang, Cheng Luo, Yiqun Liu, and Tat-Seng Chua. Temporal relational ranking for stock prediction. *ACM Transactions on Information Systems (TOIS)*, 37(2):1–30, 2019.

[4] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864, 2016.

[5] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, pages 1–14. OpenReview.net, 2017.

[6] Jong Wook Lee, Won Kyung Lee, and So Young Sohn. Graph convolutional network-based credit default prediction utilizing three types of virtual distances among borrowers. *Expert Systems with Applications*, 168:114411, 2021.

[7] Wei Li, Ruihan Bao, Keiko Harimoto, Deli Chen, Jingjing Xu, and Qi Su. Modeling the stock relation with graph network for overnight stock movement prediction. In *Proceedings of the twenty-ninth international conference on international joint conferences on artificial intelligence*, pages 4541–4547, 2021.

[8] Yi-Ting Liou, Chung-Chi Chen, Tsun-Hsien Tang, Hen-Hsen Huang, and Hsin-Hsi Chen. Finsense: an assistant system for financial journalists and investors. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, pages 882–885, 2021.

[9] Daiki Matsunaga, Toyotaro Suzumura, and Toshihiro Takahashi. Exploring graph neural networks for stock market predictions with rolling window analysis. *arXiv preprint arXiv:1909.10660*, 2019.

[10] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[11] Daniel Müllner. Modern hierarchical, agglomerative clustering algorithms. *arXiv preprint arXiv:1109.2378*, 2011.

[12] László Nagy and Mihaly Ormos. Review of global industry classification. In *Proceedings of 32nd European Conference on Modelling and Simulatio (ECMS)*, pages 66–73, 2018.

[13] Douglas Reynolds. Gaussian mixture models. In Stan Z. Li and Anil K. Jain, editors, *Encyclopedia of Biometrics*, pages 659–663. Springer US, Boston, MA, 2009.

[14] T Konstantin Rusch, Michael M Bronstein, and Siddhartha Mishra. A survey on oversmoothing in graph neural networks. *arXiv preprint arXiv:2303.10993*, 2023.

[15] Frank Switzer. CPP investments net assets total $575 billion at first quarter fiscal 2024. *Website www.cppinvestments.com* 2023.

[16] Dimitrios Vamvourellis, Mátě Toth, Snigdha Bhagat, Dhruv Desai, Dhagash Mehta, and Stefano Pasquali. Company similarity using large language models. *arXiv preprint arXiv:2308.08031*, 2023.

[17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6000–6010, Red Hook, NY, USA, 2017. Curran Associates Inc.

[18] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. *6th International Conference on Learning Representations*, 2017.

[19] Jianian Wang, Sheng Zhang, Yanghua Xiao, and Rui Song. A review on graph neural network methods in financial applications. *arXiv preprint arXiv:2111.15367*, 2021.

[20] Wonsuk Yoo, Robert Mayberry, Sejong Bae, Karan Singh, Qinghua Peter He, and James W Lillard Jr. A study of effects of multicollinearity in the multivariable analysis. *International journal of applied science and technology*, 4(5):9–19, 2014.

[21] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI open*, 1:57–81, 2020.