Deep Learning with Graphs | UE22AM342BA2

Axe Capital Investments

Assignment 2

Namita Achyuthan	PES1UG22AM100
Shusrith S	PES1UG22AM155
Siddhi Zanwar	PES1UG22AM161

Introduction

This assignment is all about learning how to leverage hypergraphs with temporal data using a bunch of different approaches- hypergraph neural networks, graph attention transformers, and even traditional machine learning methods. We've been given a stock hypergraph with data ranging from 2019 to 2022, and the task is to predict how 20 different stocks will behave in 2023.

The idea is to go beyond just individual stock trends and look at the bigger picture- how stocks are connected, how those relationships evolve over time, and how we can use those patterns to make better predictions.

Dataset Overview and Preprocessing

We began our exploration by examining the four files provided: train_stock_data.csv, validation_stock_data.csv, hyperedges.json, and bling_test_cases.json. Our first step was to understand the structure and dimensions of each file. Upon inspecting their shapes, we found that the training and validation datasets contained 20,160 and 2,000 entries, respectively. The hyperedges.json file described eight hyperedges -whose meaning we aimed to interpret- and there were 120 test cases outlined in the bling_test_cases.json file.

```
Training data shape: (20160, 102)
Validation data shape: (2000, 102)
Number of hyperedges: 8
Number of test cases: 120
```

A further look into the training dataset revealed 20 unique tickers, which we identified via the distinct values in the 'Ticker' column. These tickers were essential for mapping out the underlying structure of the data.

```
Available tickers in training data: ['AAPL' 'AMZN' 'BA' 'BAC' 'C' 'CAT' 'CVX' 'DUK' 'GOOGL' 'JNJ' 'JPM' 'KO'
 'MRK' 'MSFT' 'PFE' 'PG' 'T' 'VZ' 'WMT' 'XOM']
```

One intriguing observation was the presence of 102 columns in the dataset. This immediately stood out, and upon further inspection, we realized that each ticker was represented using five attributes: Open, Close, Volume, High, and Low.

```
Training data columns:
Index(['Date', 'Ticker', 'Open', 'Open.1', 'Open.2', 'Open.3', 'Open.4',
       'Open.5', 'Open.6', 'Open.7',
       ...
       'Volume.10', 'Volume.11', 'Volume.12', 'Volume.13', 'Volume.14',
       'Volume.15', 'Volume.16', 'Volume.17', 'Volume.18', 'Volume.19'],
      dtype='object', length=102)
```

	Date	Ticker	Open	Open.1	Open.2	Open.3	Open.4	Open.5	Open.6	Open.7	...	Volume.10	Volume.11	Volume.12
0	2019-01-02	AAPL	36.944462	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN
1	2019-01-03	AAPL	34.342203	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN
2	2019-01-04	AAPL	34.473390	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN
3	2019-01-07	AAPL	35.468021	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN
4	2019-01-08	AAPL	35.673153	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
11083	2022-12-23	XOM	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	11539400.0	NaN	NaN
11084	2022-12-27	XOM	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	11962100.0	NaN	NaN
11085	2022-12-28	XOM	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	10702100.0	NaN	NaN
11086	2022-12-29	XOM	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	10534000.0	NaN	NaN
11087	2022-12-30	XOM	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	11799600.0	NaN	NaN

20160 rows × 102 columns

So clearly, it seems to be a very sparse representation- but why was this of the form {attribute}.{number}? These tickers turned out to be the numbers themselves.

To simplify our understanding, we started by decoding the hypergraph and assigning numerical identifiers to each ticker. This mapping allowed for easier referencing and future processing.

```
Hyperedge 'Tech': ['AAPL', 'GOOGL', 'MSFT', 'AMZN'] -> [0, 8, 13, 1]
Hyperedge 'Finance': ['JPM', 'BAC', 'C'] -> [10, 3, 4]
Hyperedge 'Healthcare': ['JNJ', 'PFE', 'MRK'] -> [9, 14, 12]
Hyperedge 'Energy': ['XOM', 'CVX'] -> [19, 6]
Hyperedge 'Consumer': ['WMT', 'PG', 'KO'] -> [18, 15, 11]
Hyperedge 'Industrials': ['BA', 'CAT'] -> [2, 5]
Hyperedge 'Communications': ['VZ', 'T'] -> [17, 16]
Hyperedge 'Utilities': ['DUK'] -> [7]
```

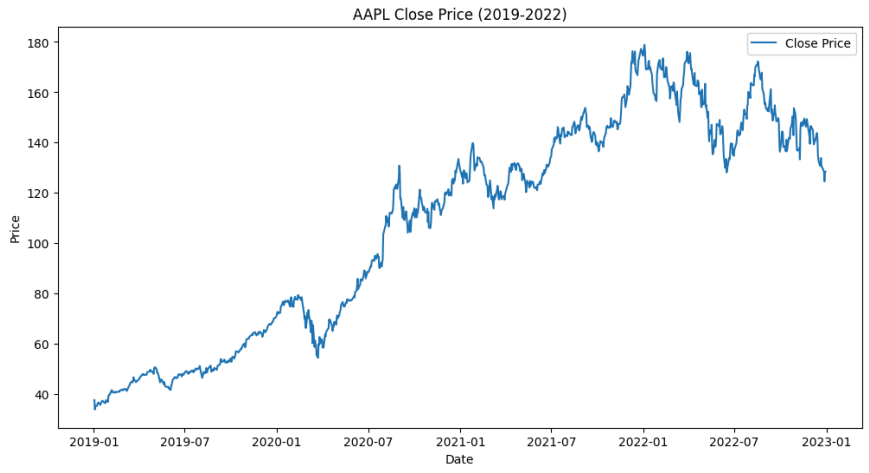
Another issue we encountered early on was with the date format- it was inconsistent and needed reformatting. We resolved this by applying Pandas' to_datetime function and then sorting the values by both ticker and date.

With the dataset now better understood, we recognized the need for a more structured and meaningful representation. The sparsity of the original format limited its utility, so we decided to create a new dataframe. In this new format, we constructed rows with seven clear columns per datapoint: *Date*, *Ticker*, *Open*, *High*, *Low*, *Close*, and *Volume*. This transformation resulted in a more conventional time-series representation, which significantly enhanced interpretability.

	Date	Ticker	Open	High	Low	Close	Volume
0	2019-01-02	AAPL	36.944462	37.889005	36.787037	37.667179	148158800.0
1	2019-01-03	AAPL	34.342203	34.757230	33.869933	33.915253	365248800.0
2	2019-01-04	AAPL	34.473390	35.432244	34.299271	35.363071	234428400.0
3	2019-01-07	AAPL	35.468021	35.499030	34.800162	35.284359	219111200.0
4	2019-01-08	AAPL	35.673153	36.212212	35.425093	35.956993	164101200.0
...	...	...	...	...	...	...	...
20155	2022-12-23	XOM	99.121032	100.780001	99.074697	100.724396	11539400.0
20156	2022-12-27	XOM	101.271219	102.383381	100.863433	102.123878	11962100.0
20157	2022-12-28	XOM	101.864365	101.947773	100.001499	100.446358	10702100.0
20158	2022-12-29	XOM	100.084912	101.688276	100.084912	101.206337	10534000.0
20159	2022-12-30	XOM	100.659532	102.411179	100.659532	102.225822	11799600.0

20160 rows × 7 columns

Following this, we began visualizing the data. We generated plots for each ticker's attributes over time. Here is an example of one such graph:



This revealed large variations across companies, prompting us to consider normalization techniques as part of our preprocessing strategy. For our experiments, we applied three primary scaling methods:

1. Standard Scaler: Standardizes features by removing the mean and scaling to unit variance.
2. Min-Max Scaler: Scales each feature to a given range, typically between 0 and 1, which can be more robust to outliers.
3. Robust Scaler: Uses the median and interquartile range, making it less sensitive to outliers and skewed distributions.

At this stage, we had covered the foundational preprocessing steps. This marks the point where our approaches began to diverge for downstream experimentation.

Modelling Approaches

Graph Attention Transformers

1. What is a GAT?

It is a type of neural network designed to work with graph-structured data. It improves upon traditional Graph Neural Networks (GNNs) by using an attention mechanism- a way to weigh the importance of different nodes (or neighbors) when aggregating information.

- In a graph, each node passes messages to its neighbors.
- A GAT doesn't treat all neighbors equally- it learns which neighbors are more important and gives them more weight.

2. Why did we choose GAT?

In our case, we're working with a stock hypergraph- which naturally involves many interconnected nodes (stocks), and the influence of one stock on another isn't uniform or constant.

We chose GAT because:

- It's well-suited for heterogeneous relationships- some stocks may have stronger influence over others at certain times.
- The attention mechanism helps focus on more relevant connections, improving prediction accuracy.

3. How was it implemented?

The model is implemented using PyTorch Geometric. It consists of:

- Two stacked GATConv layers:
  - The first GATConv takes in input node features and applies attention-based message passing across neighbors.
  - The second GATConv layer refines the learned node embeddings.
- Followed by a ReLU activation.
- Then, a global mean pooling layer aggregates the node-level information into a graph-level representation.
- Finally, a fully connected linear layer performs classification based on the pooled graph embeddings.

This setup enables the model to learn both local node-wise relationships and a holistic understanding of the graph.

4. Temporal handling and attention mechanisms

Attention is a core component here:

- The Graph Attention Network uses self-attention within the graph structure.
- Each node learns to weigh its neighbors differently based on learned attention coefficients, allowing the model to focus more on informative neighbors during aggregation.

This localized attention mechanism enhances the expressiveness of the GNN without introducing complex temporal dependencies.

Hypergraph Neural Networks

1. What is a HGNN?

A Hypergraph Neural Network (HGNN) is a type of graph neural network designed to work with hypergraphs- graphs where a single hyperedge can connect more than two nodes (unlike normal graphs which connect only two at a time). This makes HGNNs especially useful for modeling higher-order relationships among entities.

In this context, stocks are the nodes, and hyperedges are industries (Like Tech and Healthcare), connecting all stocks in that industry- representing sector-wise relationships.

2. Why did we choose HGNN?

We used HGNN because:

- Stocks don't operate in isolation- companies in the same industry tend to be influenced by similar economic, political, or consumer trends.
- A standard GNN would only model pairwise relationships. HGNN allows us to model multi-node (multi-stock) interactions via shared hyperedges (basically the industry groups).
- This provides richer contextual learning, capturing sector-level co-movement and dependencies which are vital in financial prediction tasks.

Our incidence matrix shows that every industry can share information within itself via the same hyperedge. This is exactly what the HGNN excels at.

3. How was it implemented?

In this model, stocks are treated as nodes, while industries form hyperedges, enabling the model to capture higher-order interactions that traditional pairwise graphs often overlook. The key components are as follows:

- Data structure preparation:
  - Nodes represent individual stocks
  - Hyperedges are constructed using industry information, where each hyperedge connects all stocks within the same sector. This allows the model to reason about group dynamics rather than just pairwise relationships.
  - An incidence matrix is built using hyperedge_index, efficiently encoding the multi-node connectivity structure.
- Graph snapshots (for the temporal view):
  - For each trading date, a snapshot of the hypergraph is created:
    - Node features x_t: selected stock indicators (features like returns, volatility, etc.)
    - Node targets y_t: actual prices (Open, High, Low, Close, Volume)
    - These are wrapped into torch_geometric.data.Data objects with the same hyperedge_index across time.
- Model architecture:
  - The hypergraph implementation boils down to two stacked hypergraph convolutional layers:
    - The first layer transforms raw features into a rich, non-linear representation using ELU activation

        - The second layer outputs continuous values corresponding to the five price attributes
- Training setup:
    - The model is trained using Mean Squared Error (MSE), optimized with Adam, and enhanced via learning rate scheduling for steady convergence
    - For the performance metrics, we looked at Performance metrics include:
        - Root mean squared error (RMSE)
        - Mean absolute percentage error (MAPE)

4. Temporal handling and attention mechanisms

The model *does not explicitly use temporal layers* (like RNNs, LSTMs, or Transformers). Instead, it handles time by creating independent static graph snapshots for each day. The dataset is effectively treated as a sequence of graphs over time, but:

- No temporal dependencies are modeled directly.
- Each snapshot is learned independently from others.

This setup captures day-wise spatial dependencies, essentially the relationships between stocks via hyperedges. HypergraphConv aggregates node information based on hyperedge memberships using uniform weights (all set to 1), without learnable attention over neighbors or hyperedges.

Classical ML Approach

1. What are the models used?

We used a RandomForestRegressor from scikit-learn for classic regression modeling. This model was chosen to predict the target variable using tabular features engineered from the hypergraph.

2. Why did we choose these models?

RandomForestRegressor was chosen because it is robust to overfitting, works well with a mix of features (including engineered ones like sector and moving averages), and is effective at handling tabular data. It allows us to leverage the features derived from the hypergraph without needing to modify the structure of a traditional ML pipeline.

3. How was it implemented?

- First, features were derived from the hypergraph, such as moving averages and sector averages.
- These features were appended to a dataframe representing the dataset.
- The dataframe was split into training and testing sets. A RandomForestRegressor model was initialized and trained using the training set.
- The model was then used to predict values on the test set.

This is a classic ensemble-based tree model architecture that uses a forest of decision trees to learn non-linear patterns in the data.

4. Temporal handling and attention mechanisms

There were no explicit attention mechanisms used. Temporal aspects were implicitly handled through feature engineering (like moving averages over time windows), rather than through recurrent or transformer-based models. This method enables some degree of temporal awareness without complex temporal modeling.

Comparative Evaluation

Metrics used for evaluation

MAPE shows the average error as a percentage, making it especially useful for graphs where node features vary widely in scale. RMSE tells you how far off your predictions are on average, but it can be skewed by large outliers.
For hypergraph-based models, MAPE offers clearer, more interpretable feedback.

Side-by-side performance comparison

Model	**RMSE**	**MAPE**	**Validation Loss**
GAT			
HGNN	6033094.6001	34.62%	0.1688
Classical ML	3.385	1.6%	

Strengths and weaknesses of each method

GAT

Strengths	**Weaknesses**
GAT allows each node to selectively focus on its most important neighbors, leading to smarter message passing.	Attention is restricted to 1-hop or 2-hop neighbors.
Combines local learning with global pooling to generate meaningful graph-level representations.	Global mean pooling may overlook hierarchical structures.
Compared to deeper or more complex GNNs, this model is efficient and converges fast.	-

HGNN

Strengths	**Weaknesses**
Captures higher-order relationships between stocks via shared sectors; more expressive than traditional GCNs.	Lacks adaptability to dynamic hyperedge structures

Snapshot-based approach is modular and easy to scale to longer time series.	No temporal data embedded within features- each day is treated independently.
MSE with Adam optimizer and learning rate scheduling ensure stable training.	Misses out on temporal dependencies that could improve predictive power in sequential patterns.

Traditional ML Models

Strengths	**Weaknesses**
Easy to implement and understand, especially with feature importance from RandomForest.	Moving averages offer limited temporal insight compared to sequence models like LSTMs or Transformers.
RandomForest handles non-linear relationships and works well with structured data.	Misses the benefits of attention in capturing dynamic feature relevance across time.
Hypergraph-based features like sector and moving averages still influence predictions.	Once features are engineered, model cannot adapt to new temporal patterns or context dynamically.

Conclusions