Enhancing Lithium Price Predictions: Unveiling the Power of Graph Neural Networks

Tangyi Qian

Abstract

In the quest for enhanced prediction accuracy, recent research has emphasized the value of algorithms that consider relationships between data samples, particularly when these samples exhibit inherent connections. Researchers have increasingly turned to advanced models like Graph Neural Networks (GNNs) to improve predictions, surpassing traditional approaches such as time-series analysis and deep learning models like LSTM and GRU.

This study proposes the application of GNN models to analyze a network of companies within the dynamic Lithium market. Our aim is to employ GNNs for predicting future Lithium prices and subsequently compare these predictions with results obtained from auto-regressive methods and other established deep learning models suited for time-series data, including LSTM. This research, situated at the intersection of finance, machine learning, and the Lithium industry, seeks to illuminate the predictive potential of these innovative methodologies. The central question is whether GNNs, with their proficiency in capturing nuanced relational dynamics, can provide fresh insights into predicting future Lithium prices.

Introduction

While LSTM models excel at predicting t+1 Lithium future prices, this research goes a step further by exploring whether the aggregation of t+1 stock price predictions from LSTM models, facilitated by Graph Neural Networks (GNN), can contribute to more accurate predictions for future Lithium prices.

In recent times, algorithms that consider intrinsic relations between data samples have gained prominence for their potential to enhance predictive accuracy. This trend extends

across various domains, including finance. Notably, recent research has focused on unraveling and leveraging intricate relations among companies in the stock market. To achieve this, advanced models like Graph Neural Networks (GNNs) have been introduced to augment predictive capabilities, outperforming traditional methods such as time-series analysis and deep learning models like LSTM and GRU.

This research embarks on the application of the GNN model to analyze a network of companies within the Lithium market. The primary objective is to predict future Lithium prices and subsequently compare these predictions to those obtained from auto-regressive methods and other established deep learning models, particularly LSTM. The study is positioned at the convergence of finance, machine learning, and the Lithium industry, aiming to unveil the predictive potential of these innovative methodologies. The central question revolves around whether GNNs, renowned for their proficiency in capturing nuanced relational dynamics, can offer fresh insights into predicting future Lithium prices.

Literature Review

1.Application of GNN in Price Prediction:

In the context of utilizing Graph Neural Networks (GNN) for price prediction and finance, we draw insights from a specific study documented in ***Temporal Relational Ranking for Stock Prediction.*** The authors of this paper were confronted with the challenge of constructing a sophisticated model to identify the most profitable stock portfolio. Their proposed approach involved the creation of a relational graph encompassing the selected stocks, with GNNs employed to facilitate aggregation. To elaborate further, their methodology commenced with training a time-series model to forecast the t+1 stock prices for each candidate stock. These forecasts were then interlinked within a graph structure, which could be established using either wiki-data relations or industrial affiliations. Ultimately, a GNN-based algorithm was deployed to decipher the ranking of returns for these candidates, a pivotal step in the process of selecting an optimal portfolio.

2.Lithium Market Overview:

Providing a comprehensive snapshot of the Lithium market, this section draws upon data sourced from Trading Economics, which offers insights into Lithium pricing based on spot rates for Lithium Carbonate (with a purity of 99.5% Li2CO3 min), particularly within the battery-grade segment traded in China. Lithium, characterized by its silver-white appearance, is a critical component in batteries used for electric vehicles and mobile devices. Lithium hydroxide, another significant derivative, is manufactured through a chemical reaction involving lithium carbonate and calcium hydroxide. Notably, key global players in lithium production include Chile, China, Australia, and Argentina, while leading importers of lithium are China, Japan, South Korea, and the United States.

In the context of my own research, I confront a challenge stemming from the limited study and understanding of the structure of the lithium market. Consequently, my research task involves the selection of foundational candidate entities. Through manual exploration, I've identified fewer than fifty companies operating across the lithium market's upstream, midstream, and downstream sectors, encompassing various regions worldwide. This constraint poses a challenge in constructing a comprehensive and informative network. As a result, the need arises to include intermediate entities that exhibit indirect relationships with the primary candidates. This, however, introduces another intricate question: how do these indirectly related entities connect with each other within the network, as an excessively sparse and incomplete network might not offer the desired level of informativeness.

Preparation

1. Initial Candidate Selection:

In the pursuit of identifying key players in the lithium market, I conducted searches through sources such as Forbes. These searches yielded a preliminary list of candidate companies. These candidates serve as the foundation for my approach to gathering information from the WikiData database. In essence, they form the initial search list, and any new entities discovered will be added to this list, akin to the principles of a Breadth-First Search (BFS) algorithm.

2. Utilizing WikiData for Information:

To explore company relations, I take the potential of WikiData, a comprehensive and well-established knowledge base for entities utilized in pervious papers. WikiData encompasses a vast repository of approximately 107,438,244 entities. My strategy involved using the BeautifulSoup library (BS4) to extract relations that I deemed relevant for each entity. A curated list of "relations" was created, incorporating noteworthy connections, such as "belongs to." For instance, I included relation "P414," the officially defined relation number representing "belongs to" in WikiData. While adhering to the references in papers, I retained most of the first-order relations proposed therein.

3. First Order vs. Multi-Order:

The relations listed in the relation database are inherently intuitive and are commonly referred to as 'first-order relations.' However, focusing solely on these first-order connections often provides insufficient information. Second-order relations, for instance, occur when both company A and company B belong to industry C. Even if A and B lack a direct connection, they are indirectly related through their common association with industry C. Consequently, it becomes essential to account for such indirect relations. Additionally, higher-order relations can exist.

To maintain simplicity in my experiment, I have chosen to exclusively consider first-order relations.

4.When the search stops:

In theory, the search process should cease if the marginal gain from exploring unfound entities remains below five percent after every 5000 searches. However, for the sake of simplification in this experiment, I halted the search after uncovering 150,000 relation entries, even though, at that point, the marginal gain from discovering new entities still exceeded 20 percent.

Experiment

1.WikiData relational list:

I have a small WikiData relational list with 10,051 entries and a large one with 150,000 entries.

In this case, I will only use the small one.

From this small list, I identified 5,697 unique companies that could be used to build my knowledge graph.

```
0                                [Q478214, Tesla Inc, True]
1                                [Q478214, Tesla Inc, True]
2                                [Q478214, Tesla Inc, True]
3                                [Q478214, Tesla Inc, True]
4                                [Q478214, Tesla Inc, True]
                                ...
10598                                [Q11379, energy, True]
10599                    [Q205805, conservation law, True]
10600                        [Q214070, physical law, True]
10601                                [Q11379, energy, True]
10602    [Q6211811, Category:Conservation of energy, True]
Length: 21206, dtype: object
5697
```

2.Node_list:

The second step is to check which companies have stock exchange data (their identical tickers) on WikiData. I need this stock data as an attribute of the graph nodes for training my GNN in the future. If I include all entities without stock exchange info in my graph (such as the small companies or those missing this info), my graph will be heterogeneous. The GNN model I will use can not handle this type of graph.

To simplify things and test the connectional effect gained from the relations in this network, I have decided to only keep the companies with stock info. This will result in a homogeneous graph.

To accomplish this, I check the value under the relation P414 for each company and create an array called stock_comp. Then, when building the node list, I examine the stock tickers of each company with available stock exchange data. Here is a slice of the node_list (length 137):

```
node_list[:10]
```

```
[[0, 'Q1046951', 'Target Corporation', [('New York Stock Exchange', 'TGT')]],
 [1, 'Q105966435', 'EVE Energy Co', [('Shenzhen Stock Exchange', '300014')]],
 [2, 'Q106238481', 'Piedmont Lithium', [('Australian Stock Exchange', 'PLL')]],
 [3, 'Q1065539', 'Jiangling Motors', [('Shenzhen Stock Exchange', '000550')]],
 [4, 'Q1069924', 'Zipcar', [('Nasdaq', 'ZIP')]],
 [5,
  'Q108541568',
  'Mineral Resources',
  [('Australian Stock Exchange', 'MIN')]],
 [6,
  'Q109464416',
  'LG Energy Solution',
  [('Korean Stock Exchange', '373220')]],
 [7,
  'Q10954805',
  'Ganfeng Lithium',
  [('Shenzhen Stock Exchange', '002460'),
   ('Hong Kong Exchanges And Clearing Ltd', '1172')]],
 [8,
  'Q109627145',
  'VULCAN ENERGY RESOURCES LIMITED',
  [('Australian Stock Exchange', 'VUL'), ('Frankfurt Stock Exchange', 'VUL')]],
 [9,
  'Q11104175',
  'Dongfeng Motor Group Company Limited',
  [('Hong Kong Exchanges And Clearing Ltd', '0489')]]]
```

3.Edge_list:

The next step is to build my edge list in the format of [edgeId, startId, endId, P (the relation identifier)]. Note that when building the node list, I used unique index numbers to identify each node and this is for convenience when building my edge list:

```
1  edge_list[:100]
```

```
[[0, 3468, 4988, 'P31'],
 [1, 3468, 1253, 'P31'],
 [2, 3468, 4513, 'P31'],
 [3, 3468, 3495, 'P31'],
 [4, 3468, 3306, 'P31'],
 [5, 3468, 5411, 'P31'],
 [6, 3468, 1619, 'P452'],
 [7, 3468, 1111, 'P452'],
 [8, 3468, 5689, 'P452'],
 [9, 3468, 1928, 'P1056'],
 [10, 3468, 2139, 'P1056'],
 [11, 3468, 1187, 'P1056'],
 [12, 3468, 920, 'P1056'],
 [13, 3468, 1766, 'P1056'],
 [14, 3468, 3199, 'P1056'],
 [15, 3468, 3279, 'P1056'],
 [16, 3468, 4212, 'P1056'],
 [17, 3468, 3975, 'P1056'],
 [18, 3468, 4214, 'P1056'],
 [19, 3468, 1331, 'P1056'],
 [20, 3468, 5271, 'P1056'],
 [21, 3468, 1703, 'P1056'],
```

Then, build the adjacency matrix for the graph based on the edge list and node list. Note that our network has multiple types of relations, so the adjacency matrix for this network is not two-dimensional, but three-dimensional. Conventionally, it is beneficial to test the connectivity and sparsity of this undirected graph. Since this research does not involve causal inference, there is no need to discuss Homophily. By using Numpy's count_nonzero() function on the adjacency matrix and dividing the result by the product of the matrix's dimensions, the pairwise graph density is calculated to be 0.0006486264207655123, indicating that the graph is very sparse on average. However, the average degree is 3.528523784447955, indicating a situation of heterogeneous degrees. Specifically, there may be some dense clusters (e.g., Tesla has many connections to other companies) but isolated vertices (e.g., EVE Energy CO.). The network is also not fully connected, but it is acceptable for using GNN. At this stage, after building the network that maintains the relationships between the nodes, we need to store the stock time series data for each node. To accomplish this, I iterate over the node list and use the EOD API to retrieve the time series stock price data for each company. We then append this data to the attribute array in the node list.

4.LSTM to fit on the t+1 stock price:

To train the LSTM model for predicting the t+1 price using a 60-day price window, it is crucial to use training data from a specific time range. In this case, the stock time series data from @January 1, 2014 to @December 31, 2021 has been selected. Any nodes that do not have data for this entire period will be excluded from the training data. It is important to note that this approach may exclude some useful nodes and could have a negative impact on a small network. However, this should not be a concern when using a larger graph.

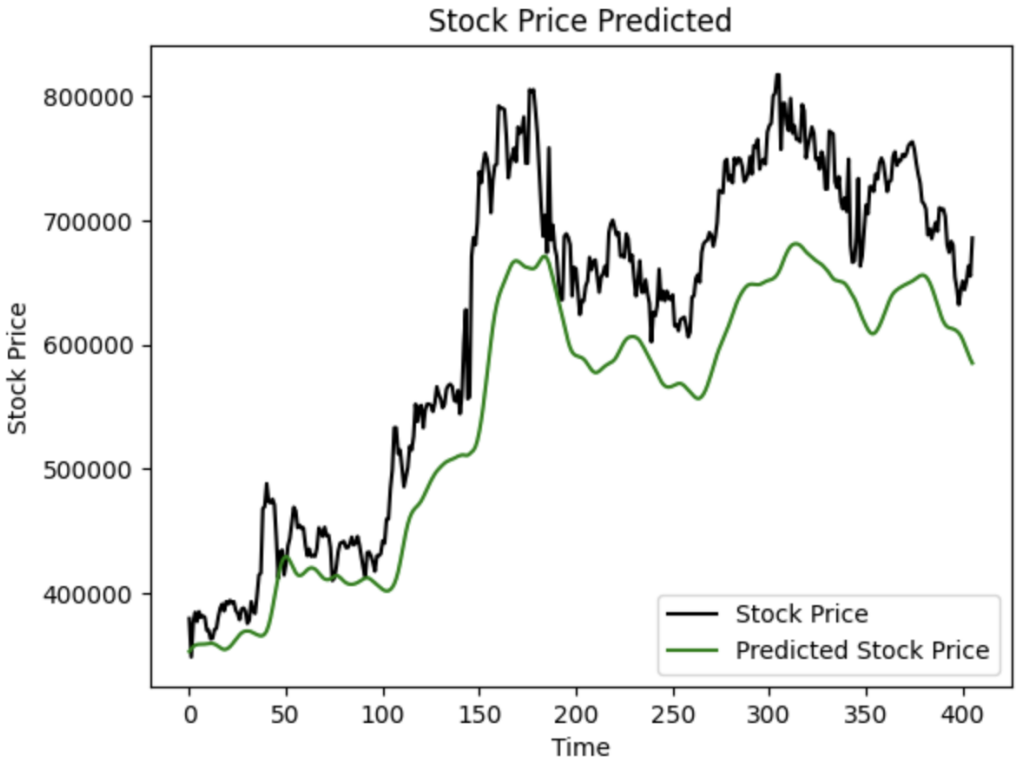
Finally, 57 nodes are left, and I refer to this remaining set as "small_node_list". I then update the "edge_list" by removing any nodes that are not included in the "small_node_list". This ensures that the final network used for GNN training contains all nodes with the required time series data. The length of the updated "edge_list" is 47.

Next, I use the "small_node_list" and "small_edge_list" to construct a homogeneous graph, which I call "small_graph". The shape of "small_graph" is 57 rows x 57 columns x 34 layers. Here, the 34 layers represent 34 types of relations. To improve training for my simple GNN, I aggregate these 34 layers into a single layer using np.sum().

I then run an LSTM model for each stock. For each stock, the training data consists of 60-day sliced windows of stock prices from 2014 to 2021. The target is the corresponding next day's price, based on the last day of the window. For example, if the window of prices is [@January 1, 2015, @January 2, 2015, ..., @March 1, 2015], the target is the price on @March 2, 2015.

The trained LSTM model represents the sequential embeddings that map the stock price sequence to the next day's price (t+1). Based on previous work, LSTM is one of the best and most convenient models for this task, so I don't want to spend time upgrading this sequential-mapping model. Additionally, when I manually plot the fitted LSTM for each stock (see the figure below), the model accurately predicts the trend of the stock price.

The decision to not use an autoregressive model like ARIMA is due to its higher mean squared error in practice, despite its fitted curve closely resembling the actual trend.



Note that when training the LSTM models for each of the stock, I assign the same LSTM structure to each of them, for computing convenience. It would be better to apply auto parameter tuning.

The learned LSTM sequential embeddings look like this: the col index represents different companies' stocks and each entry is the predicted price of stock j under date i.

	Date	1	2	3	4	5	6	7	8	9	...	48	49	5
60	2014–03–27	60.230442	37.252045	0.035016	22.681608	11.524383	51.907005	3.875131	21.119858	65.812584	...	10288.719727	0.655706	114.5129(
61	2014–03–28	60.060333	36.518463	0.034572	22.764273	11.468749	51.753651	3.783297	21.365807	65.725220	...	10304.686523	0.675546	114.3875(
62	2014–03–31	59.945469	35.865425	0.034145	22.821630	11.424247	51.596428	3.670144	21.618122	65.666779	...	10326.434570	0.697594	114.2447(
63	2014–04–01	59.902287	35.252964	0.033697	22.857033	11.392011	51.460205	3.543303	21.834419	65.652298	...	10354.645508	0.722835	114.0791(
64	2014–04–02	59.936050	34.710594	0.033221	22.881315	11.372475	51.363567	3.417797	22.009489	65.693291	...	10386.011719	0.751615	113.8963(
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
2081	2021–12–24	195.837173	109.717751	0.642859	16.677685	43.515518	49.691406	1.977950	38.593952	229.099411	...	14163.611328	18.862608	498.0197(
2082	2021–12–27	194.600113	107.971794	0.636156	16.592163	43.918961	49.861965	1.949894	38.334324	226.667557	...	14203.725586	18.772547	498.5227(
2083	2021–12–28	193.896393	106.191132	0.631248	16.502735	44.247723	50.122425	1.923139	38.109917	224.727966	...	14232.025391	18.840670	498.9529(
2084	2021–12–29	193.605057	104.613457	0.628774	16.415558	44.451698	50.448021	1.895604	37.945988	223.271271	...	14248.085938	19.011311	499.5141(

5.Setup for the final training task:

The goal of this research is to predict the future price of Lithium (evaluated by the spot price of Lithium carbonate) for time period t+1. I have access to the Lithium future price data from 2017 to 2022, provided by Trading Economics. After aligning the date range of the learned embeddings (2014-2022) with the range of the Lithium price data (2017-2022), I obtained a training dataset of 1151 days * 57 embeddings. Each entry (i,j) represents the predicted price of the jth stock on the ith day. To evaluate the model's performance, I will calculate the summed squared error (SSE) across the entire dataset. A lower SSE indicates a lower mean squared error (MSE) when given the same number of samples.

6.Deep Neural Network:

One of the baseline models is a deep neural network that predicts the future price of Lithium. After multiple rounds of parameter tuning, I obtained a model with 3 layers (128, 32, 1) that uses the Adam optimizer, 100 epochs of training, and 64 samples per batch. When evaluated on the entire dataset, the sum of squared errors (SSE) is 1995908722286.

7.Simple Graph Neural Network:

I apply a simple graph neural network to aggregate the stock prices and predict the future price at each day. The term "simple" implies that I do not use convolution when aggregating the attributes of each node. Instead, I aggregate the attributes (57 predicted prices) by multiplying the learning weight matrix X with the prepared adjacency matrix A (referred to as "small_adj"), divided by the degree matrix. This resulting matrix "XA" is then fed into a fully connected model to fit the target. The sum of squared errors (SSE) on the entire dataset is 1572595800000, which is 21.2% less than the result obtained from the deep neural network.

8.Aggregate the graph without the relations:

To test whether the performance of the prediction is enhanced by the relations, I replaced the adjacency matrix of the original graph with an identical matrix. This means that all the companies have no shared relational information, and the final prediction is solely based on the sequential embeddings. As expected, the sum of squared errors (SSE) on the whole dataset is 1841355600000, which is 18.7% higher than the SSE given by a simple graph neural network.

9.Another Baseline model, ARIMA:

Fit ARIMA models with rolling window 30, 60, and 100 (p=10, q=0, d=1), I find the SSE given by simple GNN is 99.9% less than that given by ARIMA.

```
#eva on the whole set with length 1121
print(len(trues))
print(np.sum((forecasted_values - trues)**2))
# print(- (np.sum((y[30:]-propagate(X[30:], small_adj))**2) - np.sum((forecasted_values - trues)**2)))
#LSTM: 6030187345189175.0
#      4891445225162163.0
#ARIMA:5801896758931163.0
#GNN:     1398111200000.0
#Its been proved that the GNN method has the least MSE and SSE, compared to DNN and ARIMA
print((5801896703914505 - 1572595800000)/5801896703914505)
```

```
1121
5801896703914505.0
0.9997289514308417
```

Discussion

1.Result Interpretation:

It has been proven that even if the graph is not very informative, the prediction of future Lithium prices can still be significantly improved by considering relational information. In our case, the homogeneous graph only contains 57 nodes, while the original node list is approximately 5000 nodes. If I can find a way to aggregate the relational information of these 4900 companies, the final GNN model may be further enhanced. Additionally, the original list of WikiData relations I used to obtain these 5000 companies is quite small, with only 10051 entries. On the other hand, I have a much larger relational list containing 150031 relations, which is 14 times larger than the one I currently use. Utilizing this larger relational list will result in a much larger and more informative homogeneous graph during training.

I anticipate that there will be a threshold where the marginal gain from additional WikiData relations becomes minimal.

2.Method Limitations:

In addition to the information loss when choosing to build the homogeneous graph, the Graph Neural Network model itself has a lot of room for improvement. To accelerate training, I only applied the non-convolutional graph neural network algorithm, which involves using matrix multiplication on the adjacency matrix.

Using convolution allows for better aggregation of information compared to not using it. Convolutional operations help capture local patterns and relationships within the graph structure, enabling the model to consider the neighborhood of each node when making predictions. This local information aggregation is important because it allows the model to incorporate more context and capture dependencies between nodes that are closer in proximity. Especially in this case, the network has potential strong clustering effect (very low density in contrast of average degrees), the Convolutional operation may enhance the performance of the model a lot.

3.Practical Applications: Discuss the potential applications and impact of this research in the real Lithium market:

To improve the model, I plan to propose a method for dealing with companies that only serve as intermediaries in the network and do not have stock information. This involves finding a way to handle the heterogeneous graph. I will then feed the model with a large WikiData relational list and apply the Convolutional Graph Network.

For practical use, once the model performs well in forecasting tasks, I will propose a trading strategy that focuses on buying stocks at low prices and selling them at high prices. I will test the theoretical return by trading Lithium futures. Additionally, the model can be applied to study other markets.