Stock Classification from High Frequency Market Data

Sofiane Ezzehi[1] **and Bastien Le Chenadec**[1]

[1]École des Ponts ParisTech, Master MVA

Contribution Statement

1 Introduction

The goal of this challenge is to predict the stock corresponding to a given order book. Each sample is a chronological sequence of 100 events of orders for a given stock. To make this task more challenging, a lot of the data is missing, and some properties have been normalized.

The first part of this challenge was devoluted to correctly dealing with the temporal and categorical aspects of the data. Then we focused on dealing with the change of distribution of the data between the training and the test set.

In this report, we describe the data, the different models we used, the training procedure as well as the results we obtained.

2 Data overview

Each sample in the dataset is constituted of 100 events of orders for a given stock. There are 24 different stocks which are equally distributed in the training, validation, and test sets. There are 160800 samples in the training set and 80600 samples in the test set.

Feature	Type	Description
Venue	Categorical	The venue where the order was placed.
Order id	Integer	A unique identifier, which can be used to retrace updates to the order.
Action	Categorical	The type of action (new, delete, update).
Side	Categorical	The side of the order (buy, sell).
Price	Float	The price of the order.
Bid	Float	The best buying price for the stock.
Ask	Float	The best selling price for the stock.
Bid size	Float	The number of shares available at the best buying price.
Ask size	Float	The number of shares available at the best selling price.
Trade	Categorical	Whether a trade occured or not.
Flux	Integer	The quantity of shares for this order.

Table 1: Data description.

Each event is described by 11 features, as described in Table 1. There are 4 categorical features, 5 continuous features, one integer feature (flux) and the last integer feature (order id) also has some categorical properties as it links the different events of the same order.

2.1 Visualization

2.2 Preprocessing

2.3 Graph construction

To deal with the temporal and categorical aspects of the data, one idea is to represent the data as a graph that better represents the relationships between the different events. After a bit of trial and error, we made the following arbitrary choices to construct a graph representing a sample :

1. The graph is undirected.
2. Each event is a node in the graph.
3. Each venue corresponds to a connex component in the graph.
4. If two events happen successively at a venue, there is an edge between them.
5. If two events have the same order id, there is an edge between them.

The separation of the graph into connex components corresponding to the venues is motivated by the high frequency nature of the data. Indeed, very few actors can react with high speed to the events happening in another venue, so it makes sense to assume that the events happening in different venues are somewhat independent. Note that the order id is unique to a venue so there should be no edge between the different venues.

The features that are not encoded in the structure of the graph can be placed on the nodes and the edges.

- Node features: price, bid, ask, bid size, ask size, flux.
- Edge features: action, side, trade.

We also add the venue and a time feature to the nodes, otherwise the model would not be able to distinguish between the different venues, and would not know in which direction the time flows.

3 Method

In this section, we describe the different models we used. We experimented with three main types of models: recurrent neural networks which are well suited for sequential data, graph neural networks which exploit the graph representation of the data, and statistical models that exploit features extracted from the sequence of events. From the start we knew that our final prediction would be an ensemble of models, so we explored different types of models to maximize the diversity of the ensemble. Indeed the diversity should improve the robustness of the predictions, especially given the change of distribution between the training and the test set.

3.1 Graph Attention Networks

Graph Attention Networks (GAT) [1] have been shown to be effective in many tasks. Like other graph neural networks, GATs aggregate information from the neighbors of each node to compute its embedding. The main difference with other models is that GATs use an attention mechanism to weight the neighbors of each node. Specifically we used the improved version of GATs suggested in [2].

Let G be an undirected graph with N nodes denoted $[\![1, N]\!]$. Let d_1 be the dimension of the node embeddings, and $h_1, \dots, h_N \in \mathbb{R}^{d_1}$ be the said embeddings. Let d_2 be the dimension of the edge embeddings, and $\left\{e_{i,j} \,|1 \leq i, j \leq N\right\}$ be the said embeddings. Let $W_1 \in \mathbb{R}^{d' \times d_1}$, $W_2 \in \mathbb{R}^{d' \times d_2}$ and $a \in \mathbb{R}^{d'}$. The attention weights are :

$$w(h_i, h_j, e_{i,j}) = a^T \,\mathrm{LeakyReLU}(W_1 h_i + W_1 h_j + W_2 e_{i,j}) \quad (1)$$

The attention weights are normalized using the softmax operator :

$$\alpha_{ij} = \frac{\exp(w(h_i, h_j, e_{i,j}))}{\sum_{k \in \mathcal{N}_i} \exp(w(h_i, h_k, e_{i,k}))} \quad (2)$$

where $\mathcal{N}_i$ denotes the set of neighbors of node i in G. The embedding of node i is then computed as :

$$h_i' = \mathrm{LeakyReLU}\left(\sum_{j \in \mathcal{N}_i} \alpha_{ij} W_1 h_j\right) \quad (3)$$

In general we will use multi-head attention, with K heads, $a^{(1)}, \dots, a^{(K)} \in \mathbb{R}^{d'/K}$, $W_1^{(1)}, \dots, W_1^{(K)} \in \mathbb{R}^{d'/K \times d_1}$ and $W_2^{(1)}, \dots, W_2^{(K)} \in \mathbb{R}^{d'/K \times d_2}$:

$$h_i' = \mathrm{LeakyReLU}\left(\frac{1}{K} \sum_{k=1}^{K} \sum_{j \in \mathcal{N}_i} \alpha_{ij}^{(k)} W_1^{(k)} h_j\right) \quad (4)$$

Furthermore, we will stack multiple GAT layers to obtain a deeper model. We may also apply a multi-layer perceptron to the embeddings of the last layer to obtain a more expressive representation. This model is easily parallelizable which is useful for mini-batch training.

3.2 Other graph models

We started out with GAT models because we already had some experience with them, and they were very effective. However, we also trained other graph models to diversify our ensemble (as long as they supported edge features in the graph). Here is an exhaustive list of models that gave us acceptable results and were added to the ensemble (in no particular order) :

- Generalized GNN [3]
- Pathfinder Discovery Network [4]
- Principal Neighbourhood Aggregation [5]
- Graph transformer [6]
- General GNN [7]

References

[1] Petar Velikovi, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Píetro Lió, and Yoshua Bengio. Graph attention networks. *arXiv (Cornell University)*, 2 2018. doi: 10.17863/cam.48429. URL https://arxiv.org/pdf/1710.10903.pdf.

[2] Shaked Brody, Uri Alon, and Eran Yahav. How Attentive are Graph Attention Networks? *arXiv (Cornell University)*, 5 2021. doi: 10.48550/arxiv.2105.14491. URL https://arxiv.org/abs/2105.14491.

[3] Guohao Li, Chenxin Xiong, Ali Thabet, and Bernard Ghanem. DeeperGCN: All you need to train Deeper GCNs, 6 2020. URL https://arxiv.org/abs/2006.07739.

[4] Benedek Rozemberczki, Peter Englert, Amol Kapoor, Martin Blais, Bryan Perozzi, and Google Research. Pathfinder discovery networks for neural message passing. page 12, 2021. URL https://arxiv.org/pdf/2010.12878.pdf.

[5] Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Velikovi. Principal neighbourhood Aggregation for Graph Nets, 4 2020. URL https://arxiv.org/abs/2004.05718.

[6] Yunsheng Shi, Zhengjie Huang, Shikun Feng, Hui Zhong, Wenjin Wang, and Yu Sun. Masked Label Prediction: Unified Message Passing Model for Semi-Supervised Classification, 9 2020. URL https://arxiv.org/abs/2009.03509.

[7] Jiaxuan You, Rex Ying, and Jure Leskovec. Design space for graph neural networks, 11 2020. URL https://arxiv.org/abs/2011.08843.

Appendix