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 the challenge is to predict the stock corresponding to a snapshot of a given order book. Each sample is a chronological sequence of 100 events of orders, posted or traded, for a given stock. To make this task challenging, a lot of the usual data features have been removed by the organizers, and some particularly revealing properties, like price or best bid and ask, have been hidden by centering the data around the first event of each sample.

The first part of this challenge was devoted 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

To get a better understanding of the data, we performed an in-depth series of visualizations. The idea was to see if some features – original or derived – were sufficient to differentiate between some stocks. To do so, we made use of different types of statistical plots (boxplots, histograms...) where we hoped, at each step, to see if one or several stocks were clearly identifiable.

To test the relevance of the features that we derived, we used them to train a random forest classifier on the training set. Then we used feature importance to see if the features we derived were relevant.

Let's first take a look at some of the interesting features we derived. We need to keep in mind that the goal is to extract an ensemble of features that would be discriminative for different stocks. The following list is not exhaustive, and we will only present the most interesting features.

Bid-ask spread The bid-ask spread (difference between the best buying and selling price) is a natural feature to consider in a financial context. Since it is a measure of the liquidity of the stock, we expected it to be a particularly good indicator of the stock. We plotted, on figure 1, the boxplot of the bid-ask spread for each stock. More precisely, for each stock, we considered all the bid-ask spreads of all the observations of the stock in the training set. Furthermore, we only collected the points where a limit order was placed, updated, deleted or traded. While we can see that the spread does not seem to completely separate the stocks, it is still a good indicator of the stock with a decent variety of boxplot shapes. As we will see in the results section, the bid-ask spread was actually the most important feature in the random forest classification, thereby confirming our intuition.

Another takeaway from this plot is that most stocks are fairly liquid, with an average spread of around 0.1, which represents 10 ticks. Nevertheless, some stock are strikingly more volatile than others, with a spread distribution that can have large variances.

Volume A very straightforward feature, which is actually given in the data, is the volume of the orders. We plot on figure 13 the boxplot of Bid and Ask sizes for each stock. We can see that, as in the case of the bid-ask spread, the stocks separate decently well.

Volume is also a feature that we consider to be important and relevant, since it is another measure of liquidity.

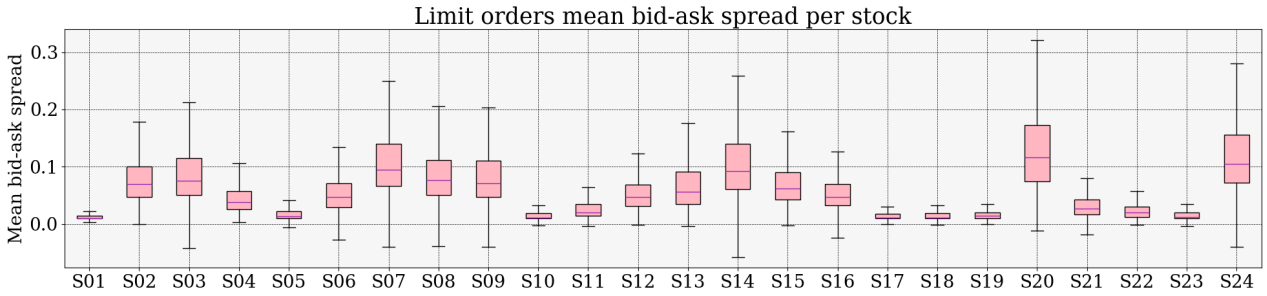


Figure 1: Boxplot of the bid-ask spread for each stock. We only considered the points where a limit order was placed, updated, deleted or traded.

Price outliers (number and price value) An *a priori* good feature we derived was the price outliers. First, we considered the number of price outliers for each stock. More precisely, given a stock, we considered all the corresponding observations in the training set, and counted the number of price outliers over the 100 events.

We define a price outlier of order i as an order whose price is i ticks away from the best buying or selling price. An illustration of this definition is given on figure 2.

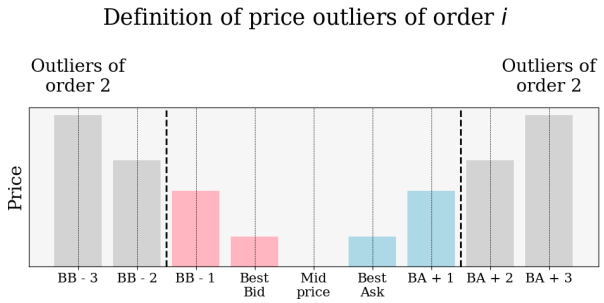


Figure 2: Illustration of the definition of price outliers. In this example, we illustrate the case of a price outlier of order 2.

While this definition may seem arbitrary since the liquidity and volatility is not the same for all stocks, it is, for our purposes, a fairly sufficient way to set a definition. This is because, the average spread is around 1 and 10 ticks for most stocks, and as we can see on figure 10, we already include more than enough outliers for the least liquid stocks (stock 20 for example).

The second feature we derived was the price value of the outliers. This is done by simply considering, for each stock and corresponding observation, the average price of the outliers over the 100 events.

We plot on figure 11 the boxplot of the number of price outliers for each stock, over all available observations in the training set. We can see that the stocks do not really separate well, and therefore that the number of price outliers does not seem to be a good indicator of the stock. This will be confirmed in the feature importance analysis in the results section.

On the other hand, we get more interesting results when we look at the value distributions of the price outliers. We plot on figure 12 the boxplot of the bid and ask price outliers for each stock. For example, we can see that stocks 7 and 20, as well as, in a lesser extent, stocks 5, 8, 15, 17, 19 and 23 distinguish themselves from the others by having a higher percentage of outliers. This type of result is what we are looking for, since it gives additional discriminative information about the stocks.

Price outliers (flux) Similarly, we looked at the flux of the orders that were price outliers. We considered, for each stock and corresponding observation, the average flux of the price outliers over the 100 events. We did this analysis on 4 different subsets of the data: ask addition, ask update, bid update and bid addition. We plot on figures 14, 15 and 16 the boxplots corresponding to the first three cases (the bid addition outliers are very similar to the ask addition outliers, so we didn't include them).

We can see that a variety of stocks are clearly distinguishable. A striking example is stock 20, which stands out alone in the ask addition outliers of order 100. Therefore, if the test set has a similar distribution, we can expect our model to perform almost perfectly on this stock.

Other unconclusive features Among the other features we looked at, we can mention the trade proportion, price and volume per stock, the bid-ask spread distinguished by venue or the proportion of limit orders. None of these features were particularly discriminative.

2.2 Preprocessing

Building on the previous section, we followed some preprocessing steps before training some models. Depending on the model, we used different pre-processing steps. We will try to describe the most important ones here.

2.2.1 Graph models

For the graph models, we used the following pre-processing steps:

1. Outliers removal : we removed all observations from the training set for which at least one event had at least one value at more than 7 standard deviations from the mean. We only considered the price, bid, ask, bid size, ask size and flux features for this operation. This removed 5808 observations which is around 3.6% of the training set.

2. Log transformation : we applied a log transformation to the flux, bid size and ask size. This specific transformation preserves the sign of the values :

$$\tilde{x} = \text{sign}(x) \log(1 + |x|)$$

3. Min-max scaling : we further normalize the bid size and ask size features by applying a min-max scaling to the log-transformed values. The min and max are computed together on both features to preserve their relative scale.

While these pre-processing steps are arbitrary, they gave much better results than some standard pre-processing steps such as standard scaling or min-max scaling on the original features.

2.2.2 *Simple features*

One set of features that we used for the statistical models and that we can refer to as "simple features" is the following:

1. For each feature present for each event (11 features), compute min, max, mean, median, and standard deviation over the 100 events.
2. Do the same for the bid-ask spread, the indicator of whether the price is the best bid or ask, and the sum of bid size and ask size.
3. Add the following features computed over the 100 events:
   - The number of unique order ids.
   - The number of orders for each venue.
   - The number of each action (new, delete, update).
   - The number orders on each side (buy, sell).
   - The number of trades.
   - The sum of the flux.

Once again these features are arbitrary, but they are a good statistical summary of the data, allowing to process a sequence of orders as a single data point in $\mathbb{R}^{89}$.

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.

[ADD A FIGURE WITH A GRAPH AND THE CORRESPONDING ORDER BOOK]

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 *Recurrent Neural Networks*

Long-Short Term Memory (LSTM) networks are a type of recurrent neural network that are well suited for sequential data [1]. We used a simple LSTM model to extract features from the sequence of events. We used the following architecture inspired by the baseline :

- An embedding layer to embed the categorical features (venue, action, side, trade) over 8 dimensions. The other features are already continuous and do not need to be embedded.
- A bidirectional LSTM layer with 128 hidden units and two layers.
- A linear layer with one hidden layer of size 128.

This model is simple and fast to train, and it gave us a good baseline to compare with the other models. We quickly reached a validation accuracy of around 50% with this model and some data preprocessing, which was encouraging. However we quickly realized that the model was not robust to the change of distribution between the training and the test set as it only reached around 30% accuracy on the test set.

3.2 Graph Attention Networks

Graph Attention Networks (GAT) [2] 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 [3].

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 :

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

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 :

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

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 $W_2^{(1)}, \dots, W_2^{(K)} \in \mathbb{R}^{d'/K \times d_2}$:

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

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.

3.3 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 [4]
- Pathfinder Discovery Network [5]
- Principal Neighbourhood Aggregation [6]
- Graph transformer [7]
- General GNN [8]

None of these models performed as well as the GAT models (although this may be due to more superficial hyperparameter tuning), but they did improve the diversity of the ensemble.

3.4 Statistical models

We used multiple statistical models on simple features extracted from the data to diversify our ensemble. These models are intuitively more robust because they rely on simple features that are less likely to be affected by the change of distribution. Here is a non exhaustive list of the models we used :

- Random Forest Classifier
- Ada Boost Classifier
- Logistic Regression
- K-Nearest Neighbors
- Ridge Classifier

3.5 Franck Zibi's model

Last year's CFM challenge also involved adapting to a change of distribution between the training and the test set. Inspired by the winning solution of last year's challenge, we devised a similar model. It is a simple model motivated by boosting methods, that learns the residuals of the predictions of a base model.

1. A random forest classifier is trained on the training set. This model is able to output a probability distribution over the classes.
2. For each class, three models are trained to predict the residuals of the random forest classifier (a random forest regressor, a k-nearest neighbors regressor and a linear regressor).
3. For each class, the three models are stacked using a linear regressor.

We can then simply predict the class of a sample by adding the output of the random forest classifier to the output of the stacked models, and taking the class with the highest probability.

3.6 Ensemble

4 TRAINING

For all the models we used the same split of the training set into a training and a validation set. We used 90% of the training set for training and 10% for validation. We did not use cross validation because the training set was already very large and well-balanced.

4.1 Loss

Like in most classification tasks, we used the cross-entropy loss to train our models. When it became clear that the distribution of the test set was very different from the training set, we experimented with different loss functions to make the models more robust to this change. We settled on Minimum Class Confusion (MCC) loss [9] which aims at minimizing the confusion between the classes on a target domain (the test set in our case).

Let $(X_n)_{1 \leq n \leq N}$ be a batch of testing samples, and $\hat{Y}_n = F(X_n) \in \mathbb{R}^{24}$ be the output of the model for sample X_n. MCC

uses temperature scaling to soften the predictions :

$$\tilde{Y}_{n,i} = \frac{\exp\left(\hat{Y}_{n,i}/T\right)}{\sum_{j=1}^{24} \exp\left(\hat{Y}_{n,j}/T\right)} \tag{5}$$

where $T > 0$ is the temperature (we used $T = 2.5$). We then compute an entropy over the predictions :

$$H_n = -\sum_{i=1}^{24} \tilde{Y}_{n,i} \log\left(\tilde{Y}_{n,i}\right) \tag{6}$$

This entropy will be used to give more importance to samples for which the model is more confident. We then use the following weights :

$$W_n = N \times \frac{1 + \exp(-H_n)}{\sum_{i=1}^{N} 1 + \exp(-H_i)} \tag{7}$$

where N is the batch size. This weight makes use of Laplace smoothing for numerical stability. Finally the class confusion between class i and class j is given by :

$$C_{i,j} = \tilde{Y}_{\cdot,i}^{T} \mathrm{diag}(W_1, \ldots, W_B) \tilde{Y}_{\cdot,j} \tag{8}$$

This class confusion is normalized to account for class imbalance :

$$\tilde{C}_{i,j} = \frac{C_{i,j}}{\sum_{k=1}^{24} C_{i,k}} \tag{9}$$

Which gives us the final loss :

$$\mathcal{L}_{\mathrm{MCC}} = \frac{1}{24} \sum_{i=1}^{24} \sum_{j=1}^{24} \left|\tilde{C}_{i,j}\right| \tag{10}$$

Note that this note does not use labels from the target domain, and is therefore unsupervised. It can simply be added to the supervised loss to make the model more robust to the change of distribution :

$$\mathcal{L} = \mathcal{L}_{\mathrm{CE}} + \mu \mathcal{L}_{\mathrm{MCC}} \tag{11}$$

we found that $\mu = 0.1$ was a good value and we could easily go as high as $\mu = 1$ or $\mu = 10$ for models that trained easily. In practice this MCC loss is computed on a batch of the test set every 10 iterations of the training loop. We found that it did raise the accuracy of the models on the test set by around 5%.

4.2 *Optimizer*

We used the Adam optimizer with a starting learning rate of 5×10^{-3}. We did not use weight decay as it hindered the training of the models.

We used a scheduler to automatically decrease the learning rate. The scheduler decreased the learning rate by a factor of 0.95 every epoch, which yielded a learning rate of around 1×10^{-3} after 30 epochs which is a typical number of epochs for our models.

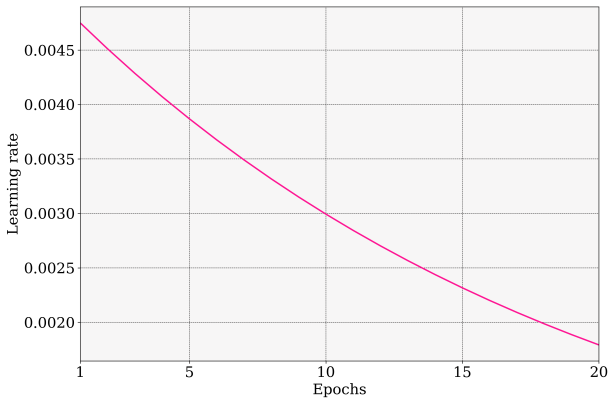


Figure 3: Learning rate schedule.

We generally stopped the training quickly when the accuracy on the validation set stopped increasing significantly. This is motivated by the change of distribution between the training and the test set, which means that we do not want to overfit the training set.

4.3 *Other unsuccessful experiments*

We tried other tricks to make the models more robust to the change of distribution.

- We experimented with data augmentation on the training set : given the high stochasticity of the data, we could easily generate new samples by adding some noise to the prices, the volumes, the flux, etc.
- Starting with a trained model, we extracted the test samples for which it was the most confident, and submitted them to the leaderboard (with the rest left at random). We could infer that we had around 80% accuracy on these samples. Thus we tried fine-tuning a model on these samples, but it did not improve the accuracy on the test set.

4.4 *Monitoring metrics*

For intelligible training and comparison of deep learning models, we monitored the following metrics at each epoch :

- Training accuracy (figure 4)
- Validation accuracy (figure 5)
- Training loss (figure 17a)
- Validation loss (figure 17b)
- MCC loss on the test set (figure 18)

These metrics allowed us to quickly stop the training when the model stopped improving, and to compare the different models. As in most classification tasks, the validation accuracy was the most important metric. Considering we used the same cross-entropy loss for all the models, the loss function can provide additional insight into the confidence of the model in the predictions.

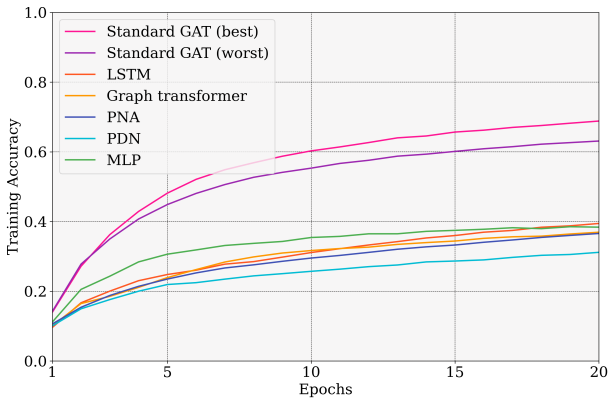


Figure 4: Training accuracy

Figures 4 and 5 provide multiple insights. First, we can see that the models are not overfitting the training set, as the training and validation accuracy are extremely close. Second, the training procedure is stable, and the models converge quickly, at least for the GAT models. The other models (LSTM, Graph transformer, PNA, PDN, MLP) might benefit from a different optimizer, but we did not have the time to experiment with this. Finally, we note that the GAT models largely outperform the other models, which is why we used them as the main models in our ensemble. The two GAT models presented are the worst one and the best one from the ensemble.

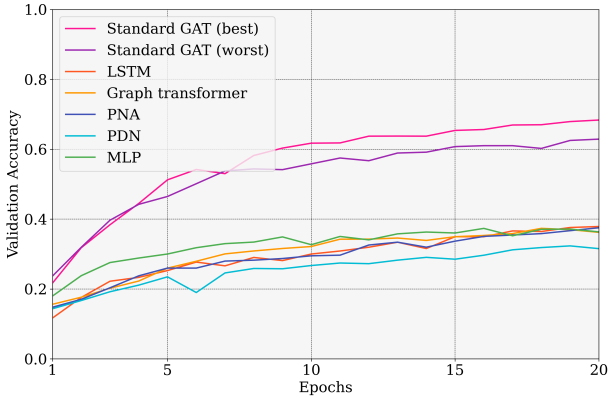


Figure 5: Validation accuracy

4.5 Model calibration

Calibration is a crucial step in the development of a classification model. It involves adjusting the output of a model to ensure that the predicted probabilities are as close as possible to the true probabilities. In other words, calibration is about making the model's confidence in its predictions accurate.

Isotonic calibration, also known as isotonic regression, is a non-parametric calibration technique. It works by fitting a piecewise constant non-decreasing function to the data, such that the predicted probabilities are as close as possible to the true probabilities.

We present in figure 6 an example of the calibration of the first class of the GAT model on the validation data. The full calibration can be found in annnex 19.

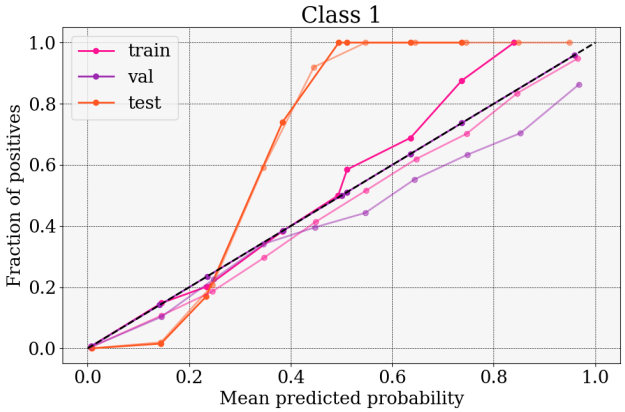


Figure 6: Calibration of the first class of the GAT model. The transparent lines represent the initial values, and the solid lines represent the calibrated values. The dashed line represents the ideal calibration. We reach an almost perfect calibration on the validation data. Note that the test data was represented using predicted class as true labels, which is why the shape of its calibration curve is meaningless. The interest of showing it is to see if the calibration affects the test predictions.

The calibration process is typically carried out on the validation data, rather than the training data. This is because the model has already been fitted to the training data, and calibrating on this data could lead to a biased result. The choice of isotonic regression over other calibration techniques is motivated by its simplicity and effectiveness, while other techniques such as Platt scaling or beta calibration delivered similar results.

In the end we did not have enough time to use calibration, but we believe that calibrating all the models before ensembling them would be a good idea.

5 RESULTS

We present our main accuracy results in table 2. The very first thing we note is the low accuracies on the test set. As we mentioned earlier, the distribution of the test set is very different from the training set, and this is reflected in the low accuracies. This leads to vastly different models on the validation set getting similar accuracies on the test set.

The GAT model gives the best results with validation accuracies greater than 70%. However it also fails to generalize to the test set. Note that the Generalized GNN model has good results but its training was not stable, so we would not trust it as much as the GAT models. The models based on features (Franck Zibi's and the MLP one) have some of the worst results, which is disappointing since we expected them to be more robust to the change of distribution.

Finally the ensemble of all the models gives us the best results with a test accuracy of 40%. We expected the ensemble to give us the best results, but we were surprised by the low accuracy, as it only gives us a 5% improvement over the best single model.

Model description	Training acc.	Validation acc.	Test acc.
PDN	0.42	0.43	0.23
MLP	0.42	0.41	0.24
Franck Zibi's model	0.95	0.42	0.25
Graph transformer	0.44	0.43	0.29
General GNN	0.43	0.42	0.30
PNA	0.47	0.45	0.30
LSTM	0.57	0.44	0.30
GAT (50 epochs)	0.75	0.71	0.33
GAT (20 epochs)	0.72	0.68	0.34
Generalized GNN	0.73	0.71	0.35
Final ensemble			0.40

Table 2: Main accuracy results

5.1 Statistical models

Random Forest Classifier Throughout our experiments, we found the random forest classifier handy for feature selection, as well as for training a model purely based on handcrafted features.

We present in this section the results of 2 random forest classifiers that we trained. The first model was trained on the Bid-Ask spread, the Bid and Ask volume, the number of price outliers, the price outliers, the flux of the price outliers, and the proportion of each venues on which the orders were placed. The second model was only trained on a subset of the features of the first model, which were the Bid-Ask spread, the Bid and Ask volume and the venue proportion.

We obtained a very important result that framed our approach for the rest of the challenge. We found that the first model had a 49% accuracy on the validation set, while the second model had a 28% accuracy. However, both models performed very similarly on the test set, with an accuracy of respectively 22% and 20%. This result is very important because it gives us a clear indication that the "outlier" features, while being very discriminative on the training set, are almost irrelevant on the test set. At this point, we had 2 choices: either we could eliminate the outlier features from our ensemble, or we could try to find a model that would be able to detect how the "outlier" features distribution changed in the test set.

5.2 Predictions analysis

After failing to improve the accuracy of our models on the test set, we looked into the predictions of the models to try to understand why they were so bad. We quickly had the idea to add noise to the input samples to see how the models would react. Thus we selected some observations for which we ran the predictions 1000 times with a white gaussian noise of mean 0 and standard deviation 0.05 added to the price, bid-ask spread, bid size, ask size and flux. We then plotted the boxplot of the predicted probabilities for each class. We present the results for the GAT model in figure 7, 8 and 9, for some selected observations from the validation set. For each figure, we show the true class, the predicted class, the predicted probabilities without perturbations, and the boxplot of the predicted probabilities for each class with the perturbations.

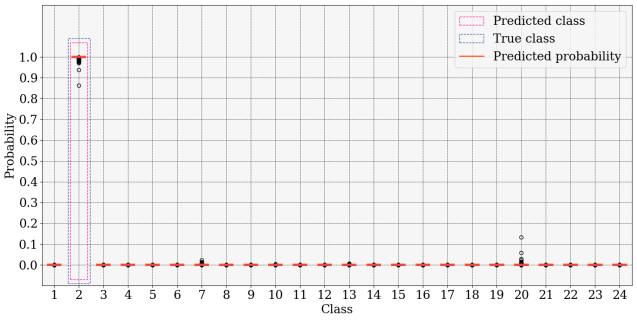


Figure 7: Boxplot of the predicted probabilities for each class.

The first example in figure 7 is the ideal case, in which the model was initially entirely confident in its prediction, and the perturbations did not change this confidence.

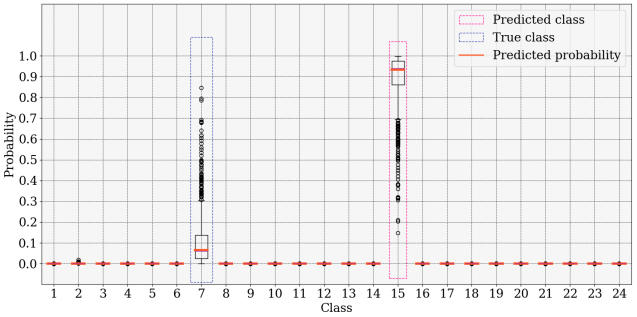


Figure 8: Boxplot of the predicted probabilities for each class.

The second example in figure 8 is the one that motivated us to look into the predictions of the models. We can see that the model was initially very confident in its prediction of the erroneous class 15, but a lot of the perturbated predictions are now in favor of the true class 7. In some of these cases it would be worth it to trust the perturbated predictions more than the initial prediction.

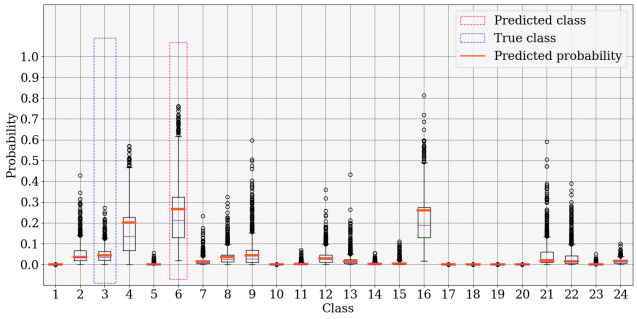


Figure 9: Boxplot of the predicted probabilities for each class.

While our two first examples were cherry-picked, in many cases the model would be very uncertain in its predictions, and the perturbations would still be very uncertain. This is the case in figure 9. Because the samples we tried were not overwhelmingly interesting cases, we did not explore this further. However note that the results on the test set were very similar to the results on the validation set, and this approache seems promising to improve the accuracy of the models.

6 CONCLUSION

REFERENCES

[1] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term memory. *Neural Computation*, 9(8):1735–1780, 11 1997. doi: 10.1162/neco.1997.9.8.1735. URL https://doi.org/10.1162/neco.1997.9.8.1735.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

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

[9] Ying Jin, Ximei Wang, Mingsheng Long, and Jianmin Wang. *Minimum class confusion for versatile domain adaptation*. 1 2020. doi: 10.1007/978-3-030-58589-1\{_}28. URL https://doi.org/10.1007/978-3-030-58589-1_28.

Appendix

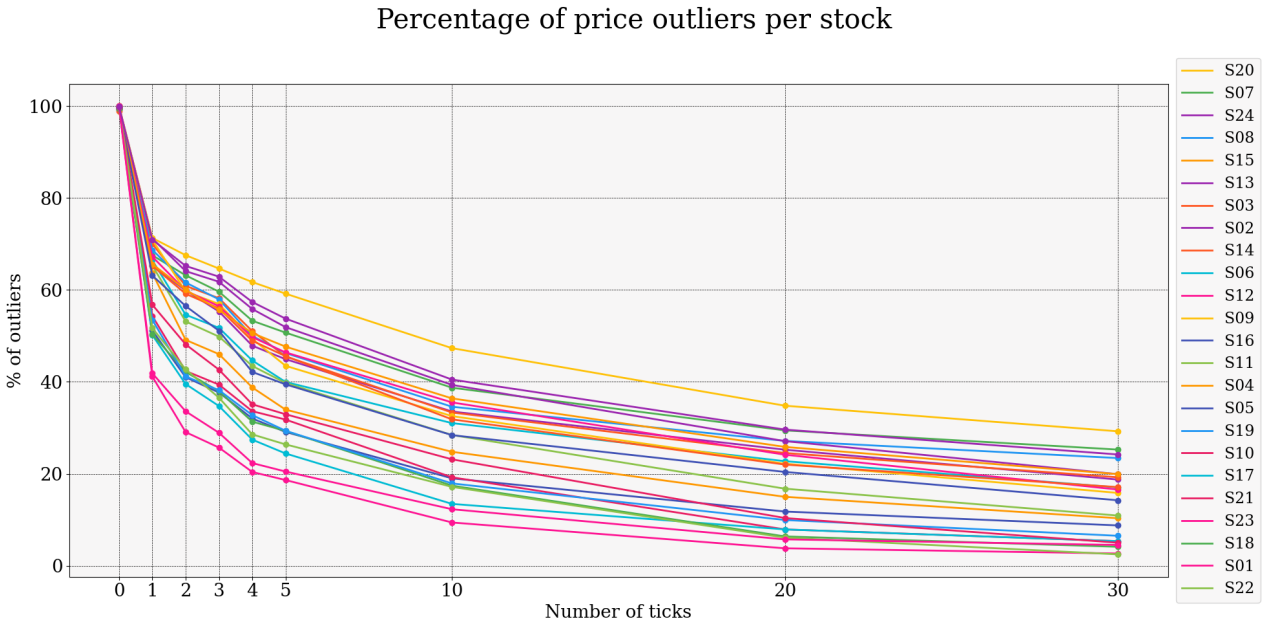


Figure 10: Percentage of price outliers per number of ticks.

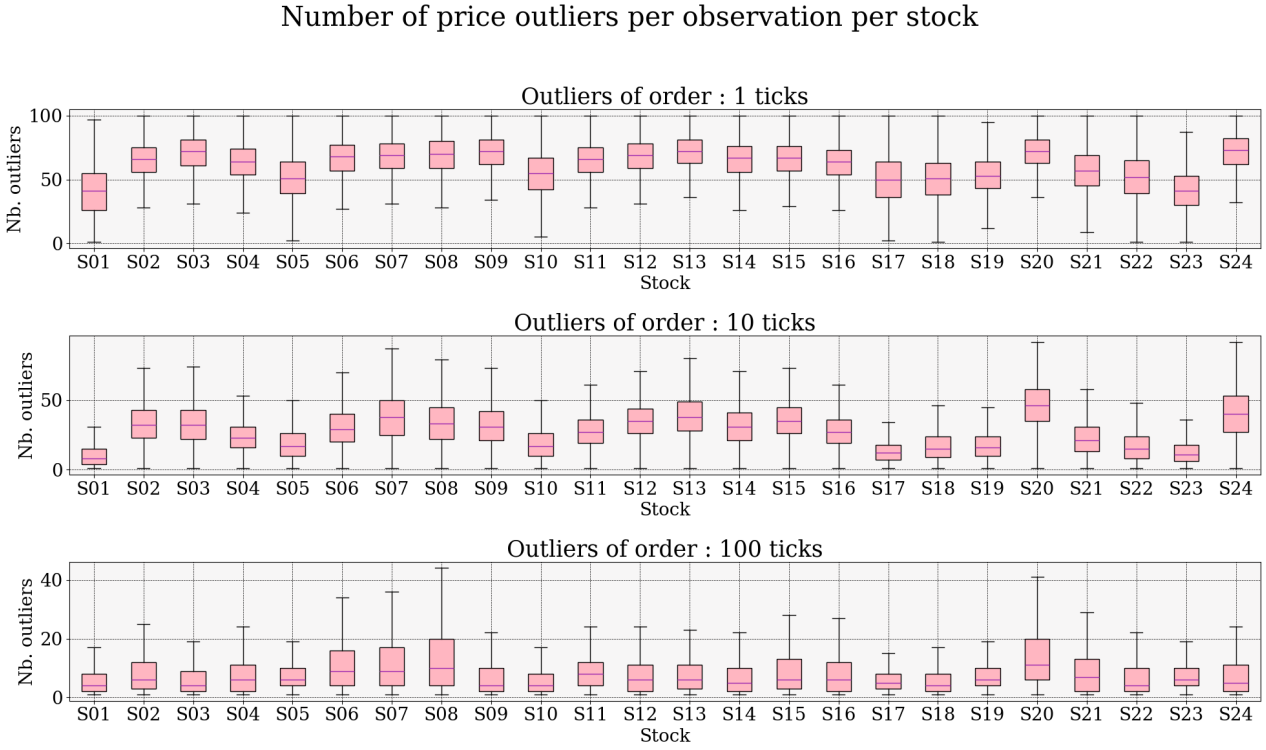


Figure 11: Boxplot of the number of price outliers for each stock.

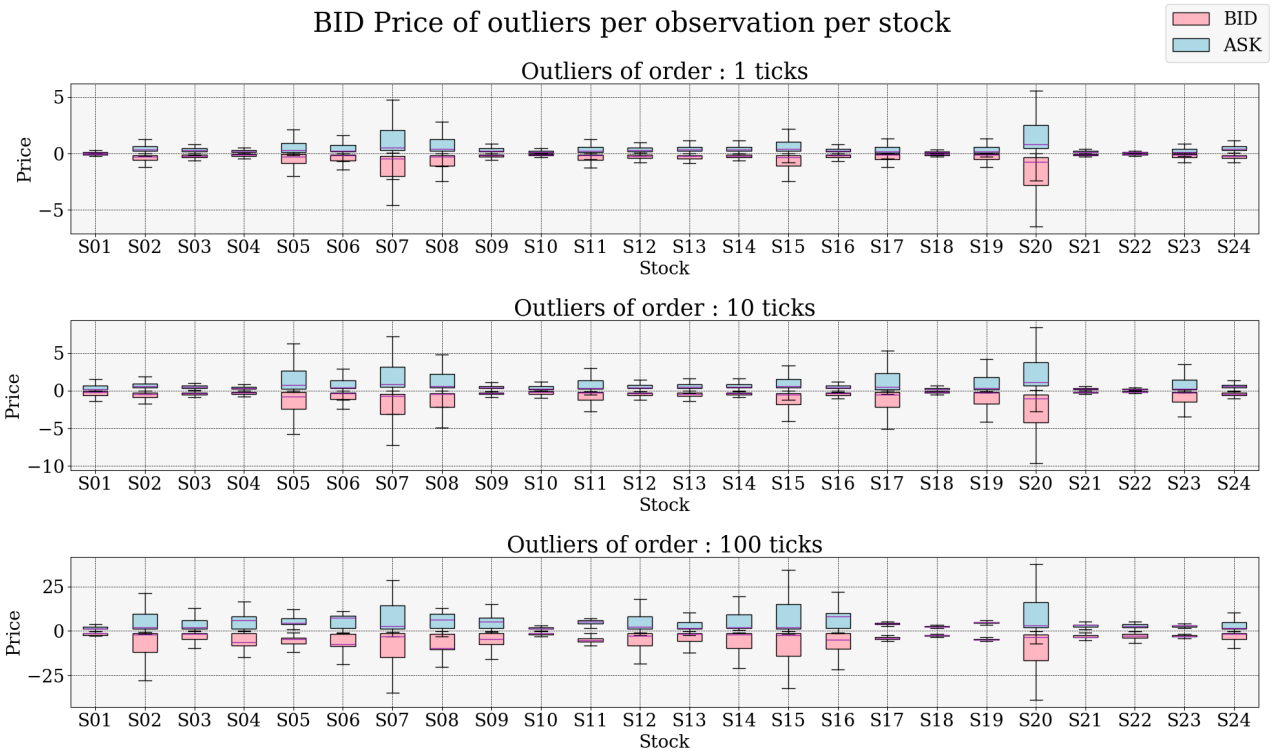


Figure 12: Boxplot of the bid and ask price outliers for each stock.

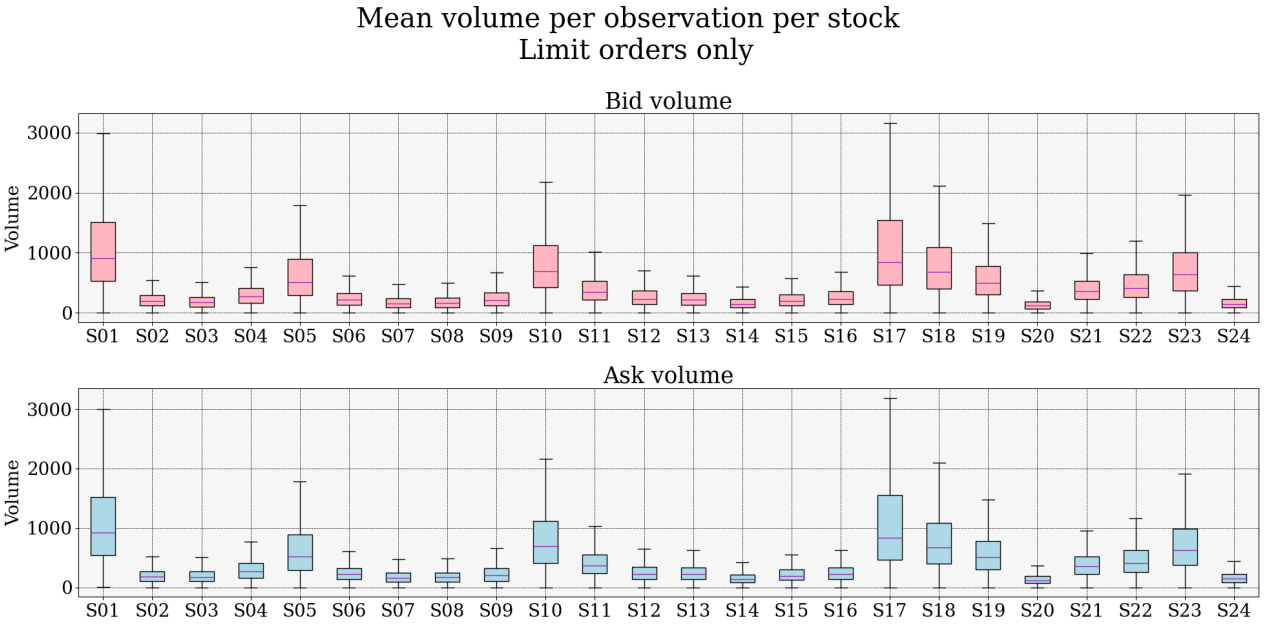


Figure 13: Boxplot of the bid and ask volume for each stock.

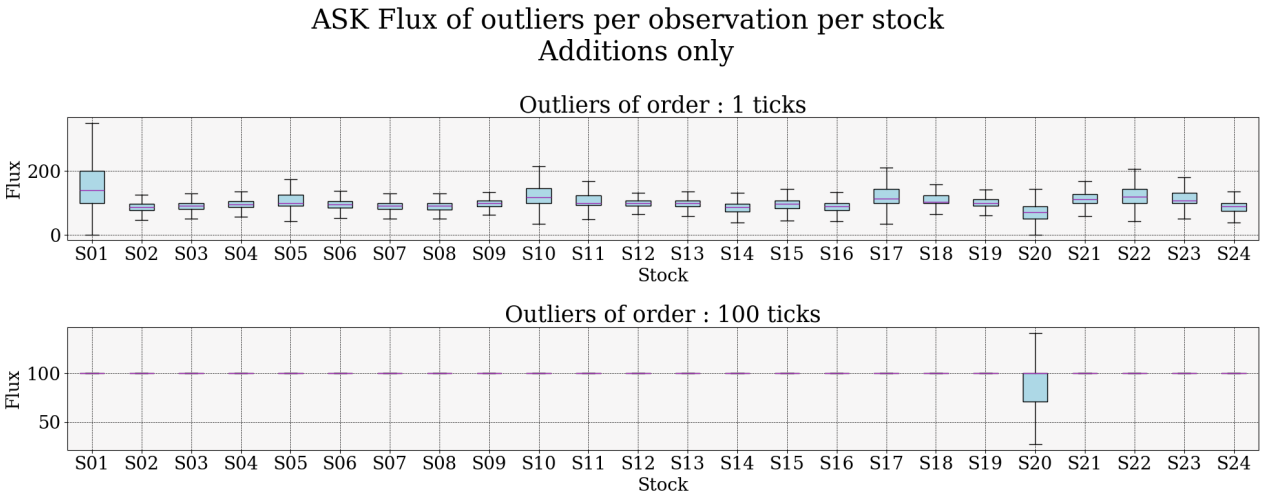


Figure 14: Boxplot of the flux of the ask price outliers for each stock. Only the ask addition outliers are considered.

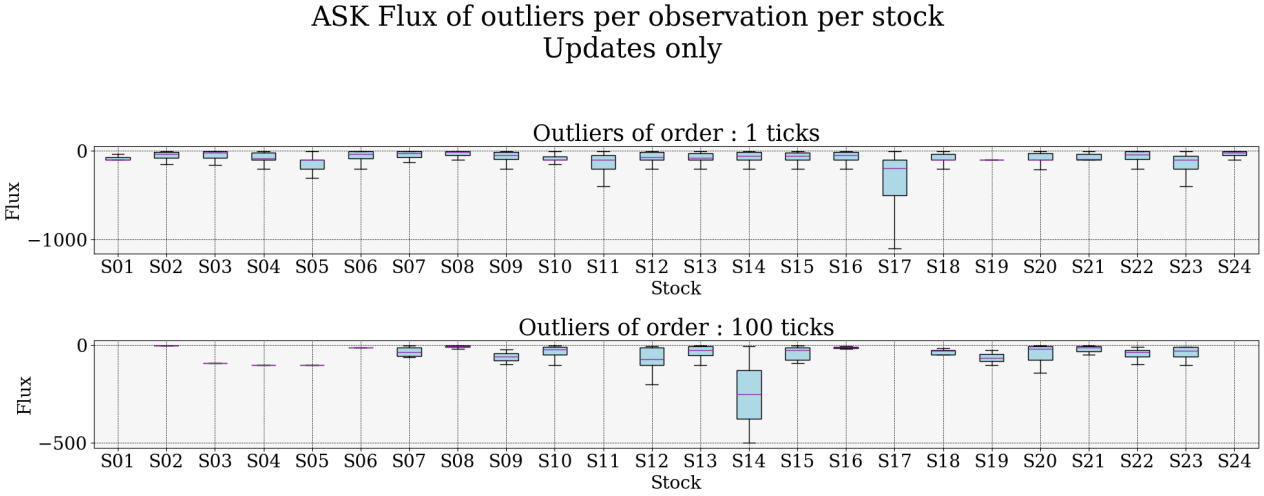


Figure 15: Boxplot of the flux of the ask price outliers for each stock. Only the ask update outliers are considered.

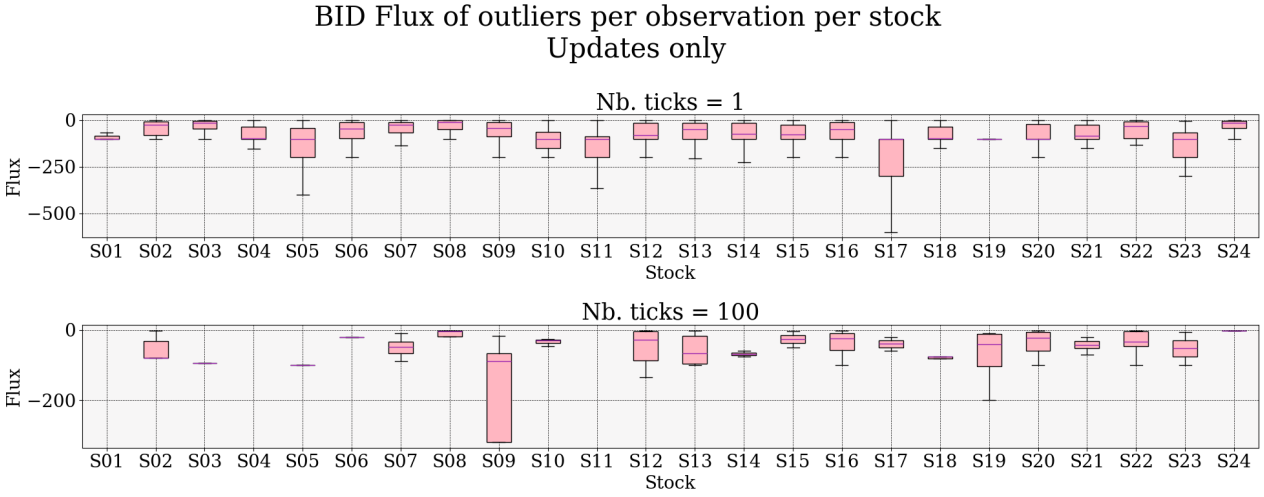
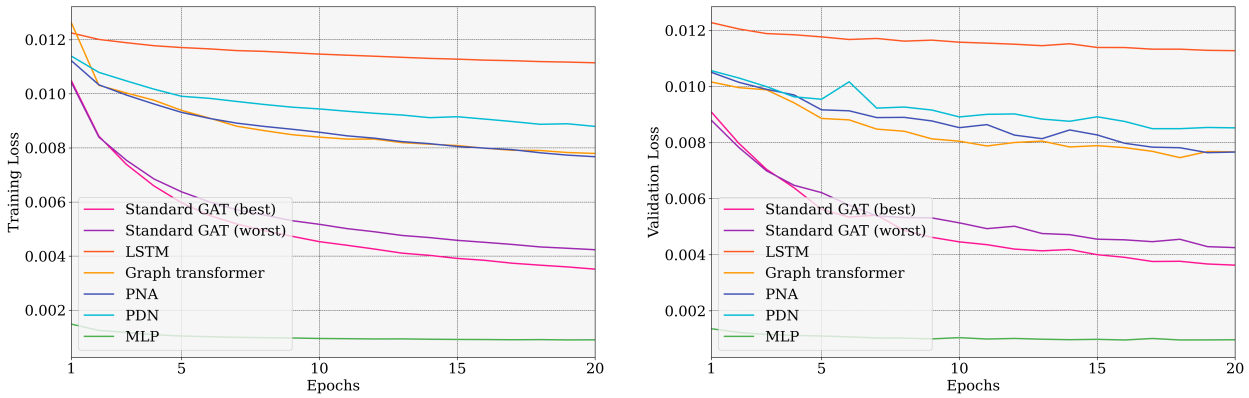


Figure 16: Boxplot of the flux of the bid price outliers for each stock. Only the bid addition updates are considered.



(a) Training loss **(b)** Validation loss

Figure 17: Training and validation loss

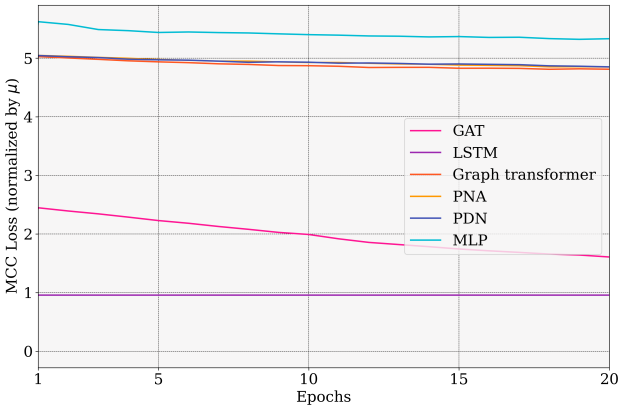


Figure 18: MCC loss on the test set

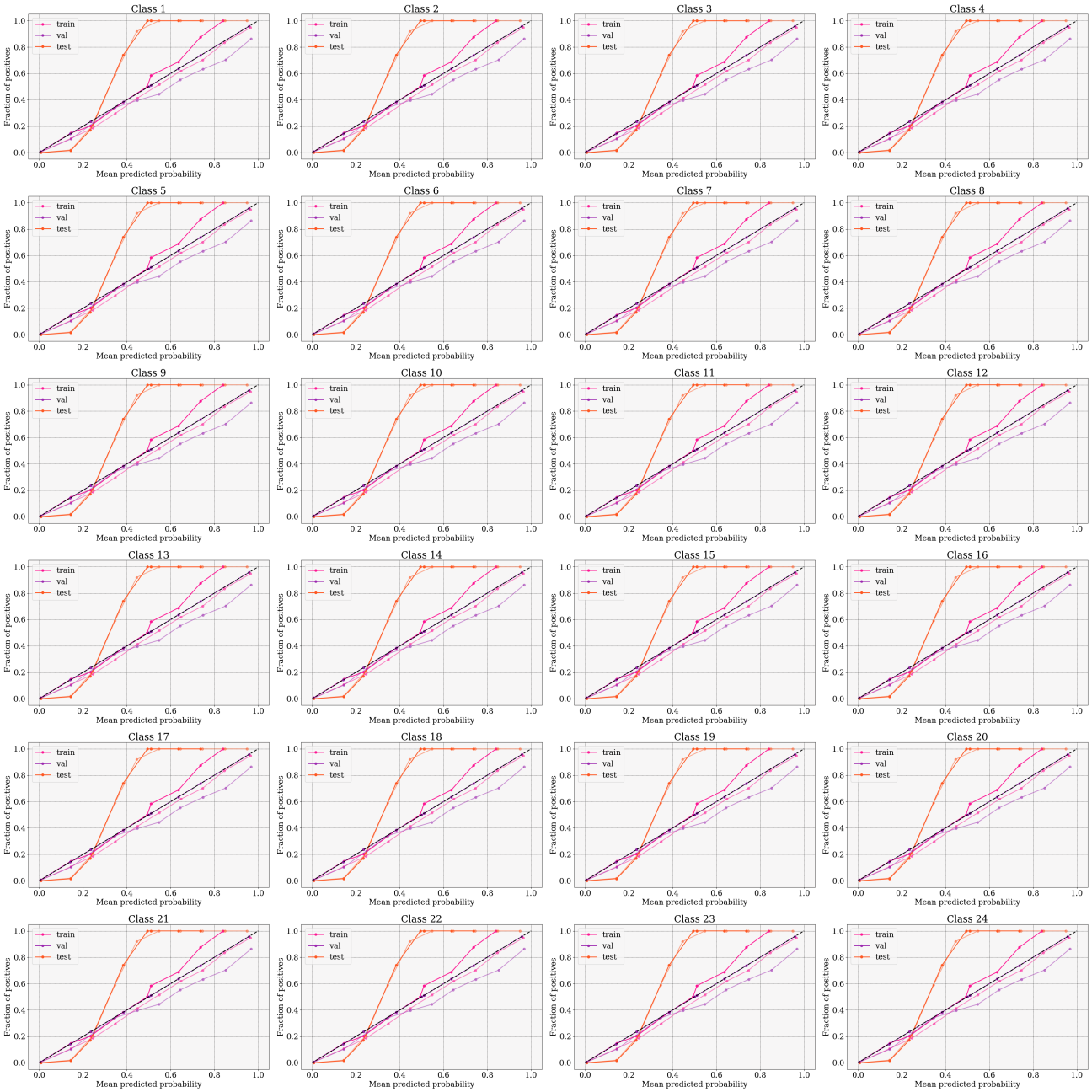


Figure 19: Calibration of the GAT model.