1 Introduction [20 points]

Group member(s): Liting Xiao (Kaggle username: Naomi Xiao).

Team name: Turtle School Son Goku.

Private leaderboard place: 39th.

AUC score: 0.62214.

Division of labor: The whole of the project, including data exploration, data handling, model building, model training, model selection, and the writing of this report.

2 Overview [20 points]

Models and techniques tried:

- Ridge regression with sigmoid activation:
Linear model with regularization was the simplest model I could try out first in this regression-like classification problem. I tried Ridge regression using different regularization strength, and after training, I used the sigmoid function to put my predicted values in the [0, 1] range. (I also tried Lasso, but L1 regularization wasn't working well.)

- Logistic regression with 5-fold cross validation:
I used *sklearn.linear_model.LogisticRegressionCV* for the 5-fold cross validation logistic regression. The runtime was long but the result was the second best, following neural networks.

- Support vector machine regressor:
sklearn.svm.SVR took forever, and I later found out that its computational complexity is $O(N^3)$. So obviously this was not the way to go with our large datasets. So as a last attempt in the SVM area, I tried *sklearn.svm.LinearSVR*, which was much faster dealing with large datasets, but the performance wasn't impressive.

- Decision trees and random forests:
These two were just a lost cause for me. They ran rather slowly and always overfitted. I tried tweaking "max_depth" and "min_samples_leaf" but couldn't really find appropriate hyperparameters' ranges. So I gave them up after a while.

- Neural networks with train/validation sets:
Multilayer perceptron worked the best for me. It wasn't too slow with shallow layers and the performance wasn't too bad after all. I tried customized neural network architectures using Google Colab's GPU resources, but they ran extremely slowly and the AUC score wasn't improving that much within 10 epochs that I managed to run within hours. Maybe it's because of the slow execution time, and maybe if I could make it run much faster, I could see improvements over tens of epochs.

Work timeline:

Not aware of the project timeline, I started out a little bit late. On the 11th, I learned about *pandas* (and *Kaggle*), explored the data structure and finished the data handling part. Also brainstormed and researched possible models for this binary classification problem that requests class probability outputs. I started implementing and trying out methods and gauging their performance on the 12th and finished it with middle-rank score. Being a complete novice to machine learning, I did not know more learning models outside of class and wasn't able to gather more novel methods for this project.

3 Approach [20 points]

Data processing and manipulation:

- Data processing:

I used *pandas* to read the *.csv* data files. With *pandas*'s various functionality, I found out that there were 26 features stored in both files (not counting "id"). And there were missing values in columns "opened_position_qty " and "closed_position_qty". Also, we could see that ranges of the these features vary a lot. So, we need to deal with missing values, feature scaling and probably feature engineering if suitable.

When building neural networks in *PyTorch*, in addition to using *pandas* to process the data, I wrote a *torch.utils.data.Dataset* class and later used *torch.utils.data.DataLoader* for easy iteration of the data.

- Data manipulation:

To deal with missing values in "opened_position_qty " and "closed_position_qty", I noticed that "transacted_qty" column was complete and that "transacted_qty" = "opened_position_qty " + "closed_position_qty". So I took "transacted_qty" for each corresponding entry, divided it by 2, and filled it to both "opened_position_qty " and "closed_position_qty". I ensured that the filled quantities were strictly integer by taking the ceiling and the floor of the half quantities if "transacted_qty" were odd numbers.

Some of the columns in the datasets were not independent of each other, for example, {"opened_position_qty " , "closed_position_qty", "transacted_qty", d_open_interest} and {"mid", "bid1", "ask1"}. But I decided not to drop any features because I verified that for the models I've used, dropping features wasn't changing or improving the performance. I added a few features that made sense to me when thinking about this problem. They were:

- **diff_price:** ask1 - bid1. How big of a gap between the lowest ask and the highest bid?
- **diff_vol:** bid1vol - ask1vol. How big of a difference between the lowest ask volume and the highest bid volume?
- **pot_vol:** ask1vol + bid1vol. What is the potential transacted quantity in the next time stamp?
- **frac_price:** last_price / mid. How big/small is the most recent order price compared to the mid price?
- **ask_spread:** ask1 / ask5. What's the spread of the ask prices?
- **ask_vol_spread:** max(ask_vols) / min(ask_vols). What's the spread of the ask volumes?
- **bid_spread:** bid1 / bid5. What's the spread of the bid prices?
- **bid_vol_spread:** max(bid_vols) / min(bid_vols). What's the spread of the bid volumes?

Finally, with missing values filled and new features added, I transformed all features into range [0, 1] by using *sklearn.preprocessing.MinMaxScaler*. And I set aside 20% of the training set to use as a validation set.

Details of models and techniques:

As explained in Section 2, I wasn't too successful with SVM or decision trees/random forests. I obtained somewhat acceptable results for the other three models tried out: Ridge regression with sigmoid activation, Logistic regression with cross validation, and neural networks. So I will explain these three models in more details in this section.

- Ridge regression with sigmoid activation:

Although this is a classification problem that outputs probabilities, Ridge regression is what I tried first since it's using a simple linear model and the continuous-valued outputs could be easily transformed into probabilities using sigmoid function. It's easy to implement using *sklearn* and it ran really fast. But the downside is that this was probably too simple a model and didn't capture the complexity of this problem.

Using *sklearn.linear_model.Ridge*, I varied the regularization strength α from 1 to 10000, and found out that $\alpha = 1$ works the best in this case.

- Logistic regression with 5-fold cross validation:

Logistic regression is a classic model to use in classification problems, so it makes sense to try it out. I tried *sklearn.linear_model.LogisticRegression* and varied the regularization strength just like in the case of Ridge, but did not see better performance. So I tried *sklearn.linear_model.LogisticRegressionCV* with 5-fold validation. Downside is that it took much longer to finish, but upside is that the performance boosted a lot.

- Neural networks with train/validation sets:

Lastly, I tried neural networks with various architectures. In particular, multi-layer perceptrons with hidden layers of size (100), (100, 50), (100, 50, 10), (256, 128, 64), (256, 128, 64). I tried it first using *PyTorch*, but it took forever to train even with one single fully-connected layer and ReLU activation. I also tried using Google Colab's GPU resources, but the execution still took forever (I still did not figure out why it took so long. The device showed 'cuda:0' but maybe I wasn't porting the computation to GPUs correctly.). So I had no choice but to resort to *sklearn.neural_network.MLPRegressor*. I saw performance boost compared to *LogisticRegressionCV*. And it had comparable execution time scale compared to *LogisticRegressionCV*. One downside of this method is that it wasn't very flexible with choosing architectures and objective and activation functions. Upside is that it was probably optimized to run fast.

4 Model Selection [20 points]

Scoring:

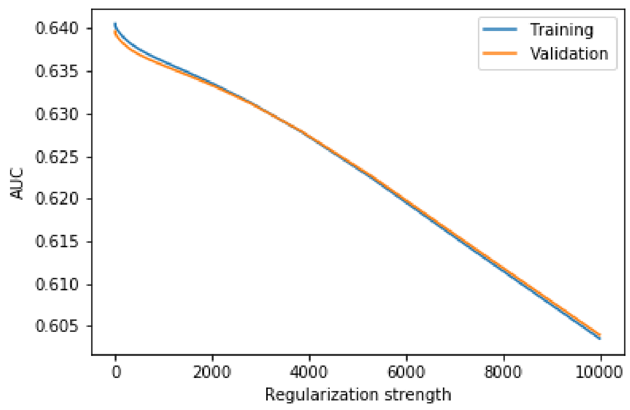
Objectives: I used L2-norm as the loss function for all three models. This choice was purely empirical. Due to the limited selection of models, the main loss function choices were L1 and L2. I tried L1-norm but it wasn't scoring well at all using these 3 models for this problem. L2 norm, on the other hand, performed well. In terms of optimizers, Ridge regression used 'auto' solver. LogisticRegressionCV used 'sag' solver. MLPRegressor used 'Adam' optimizer.

Scores: I used AUC scores on both the training and validation sets to determine the performance of a model. Simply because that's the description of the problem: we are gauging our models using AUC. It made sense, then, to use AUC as the score to check model performance.

The neural network with (100, 50) hidden layers performed the best, with (private 0.62214, public 0.62935) scores. Then the neural network with (256, 128, 64) hidden layers, with (private 0.61946, public 0.63052) scores. Then logistic regression, with (private 0.61569, public 0.61995) scores. Lastly, Ridge regression, with (private 0.58600, public 0.59360) scores.

Validation and test:

For Ridge regression, I used a set-aside validation set to check the performance of it while varying regularization strength α from 1 to 10000, as mentioned in Section 3. Here shows the AUC score vs. α. After training a Ridge model using $\alpha = 1$ and saw that the validation score was close to the training score, which indicates no sign of overfitting. I fit the model again using the whole training dataset.



For *LogisticRegressionCV*, it's already built-in to do cross validation on-the-go in *sklearn*. And I chose 5-fold cross validation here.

For *MLPRegressor*, it's also built-in that when training, the model will use 10% of the input train data to check for early stopping. But I also set aside a validation set (20% of the train set) for testing after the model's trained. And after verifying that no overfitting occurred, I re-trained the model using the whole dataset.

5 Conclusion [20 points]

- Feature importance:

By looking at the coefficients of the trained models, I found the top 10 features to be 'frac_price', 'diff_vol', 'bid1vol', 'diff', 'bid2vol', 'transacted_qty', 'opened_position_qty ', 'closed_position_qty', 'bid4vol', 'bid3vol'. Most of these features are directly related to 'mid' price or the variation of transaction volumes happened in the last 500ms.

These 9 features negatively influenced the prediction targets: 'ask1vol', 'ask_spread', 'bid_spread', 'pot_vol', 'ask2vol', 'ask5vol', 'ask3vol', 'd_open_interest', 'ask4vol'. And the other 23 features positively influenced the prediction targets.

- Why do we use AUC? Any better metric? Why or why not?:

AUC is a classic performance metric for classification problems. It measures how effective a model is at distinguishing between classes. We use AUC because it's scale invariant and we don't need to calibrate our probabilities. I think this is quite a good performance metric for this problem, since we are interested in how good the classifier is at separating classes, not at the classification errors or the uncertainty in the classifier's predictions. Also, our dataset is quite balanced with enough balanced data in both classes, which makes the use of AUC also reasonable.

- Among the machine learning methods/pipelines that your group uses, are there any methods/pipelines that are parallelizable?

I used *sklearn*'s built-in methods *Ridge*, *LogisticRegressionCV*, and *MLPRegressor* to tackle this problem. Among them, *LogisticRegressionCV* is parallelizable, with an argument 'n_jobs' to specify the number of computer cores to use. *MLPRegressor* is also probably parallelizable since it's a neural network, but I didn't find a similar argument in this method. If I managed to successfully implement a neural net in *PyTorch*, I could send the training process to GPUs for parallel operations.

- Did you learn anything new from this project outside of the lectures and homework?

Yes. I learned about *pandas*, a data exploration module I haven't used before. I learned about dealing with missing data and adding features. I briefly read about most of the learning models in *sklearn*, models we haven't learned yet in class like the 'Naive Bayes' model or the combined model 'ElasticNet'. I also learned about different performance metrics and where they are most applicable.

- Overall, what did you learn from this project?

Overall, the most important thing I took away from this project is the complete process/pipeline for such a machine learning project. From the very beginning where I need to explore the datasets at hand, to data manipulation and data processing, to model building and model selection, and at last, the performance evaluation using different evaluation metrics.

- What could you have done differently? What obstacles did you encounter during the process?

I could have engineered better features for use, and taken out the features that negatively impacted the prediction targets.

More importantly, I could have used *PyTorch* to build a neural network and sent it to GPUs for training. But the main obstacles here are in designing model architecture, which I have less experience in and in successfully sending the process to GPUs. As I explained before, my failed attempt in using *PyTorch* resulted from extremely long training time even when using GPUs and low-to-nothing performance improvements. So I suspected that a) I wasn't actually sending the models to GPUs properly, b) I didn't design an effective neural net architecture.