1 Introduction [20 points]

Group members: Emily Park, Megan Tjandrasuwita, Elizabeth Yam
Team name: Yoshi's Island
What place you got on the private leaderboard: 61
What AUC score you got on the private leaderboard: 0.60505
Division of labor: All three members collaborated on researching and choosing the appropriate model for the task. Emily and Elizabeth implemented the gradient boosting algorithm using the catboost package. Megan implemented feature extraction and a method of splitting the training data into 5 folds to use for cross validation. Through cross-validation, Megan and Emily determined the best combination of model parameters to use. Elizabeth and Emily determined the hyperparameters to set and used cross validation to find the optimal values for the learning rate of gradient descent, the number of estimators in gradient boosting, the regularization strength, and the maximum depth.
Github Repo: https://github.com/eppark/High-Frequency-Price-Prediction

2 Overview [20 points]

We used a variety of techniques over a few days to get a satisfactory model.

Models and Techniques

We mostly used gradient boosting algorithms such as CatBoost with cross-validation and implemented feature extraction methods. We tested out CatBoost regressor and CatBoost classifier, with and without feature extraction for both. We also compared the different hyperparameter values for each. We eventually settled on using gradient boost classifier with our feature extracted method without cross-validation, since this gave us the best testing accuracy.

Work Timeline

- Saturday and Sunday: Researched successful models (e.g., as CNN, NN, Boosting) and key features (e.g., bid-ask spread) in predicting Limit Order Books (LOB).
- Tuesday: Implemented CatBoosting method and key feature extraction. Fine-tuned CatBoosting model and compared hyperparameters.
- Wednesday: Tested CatBoosting Regressor vs. CatBoosting Classifier. Fine-tuned CatBoosting model.
- Thursday: Tested different CatBoosting Classifier models with different inputs.

3 Approach [20 points]

We tried a variety of approaches until we found one that worked best.

Data Processing and Manipulation

Without any data manipulation, we used the basic gradient boosting regressor with CatBoost to get a training error of 0.141 with MSE and a testing accuracy of around 0.60505, computed using AUC. Thus, we decided to implement some feature extraction methods since using the raw data to train did not represent the predictions of the mid prices well enough.

With some research, we discovered that the spreads of the asks to bids, the average price/volume values at each level in the order book, and the accumulated price/volume differences were some good features to extract. We contemplated using an autoencoder to extract hidden features and reduce dimensionality, but instead decided to implement feature extraction by hand to gain a better understanding of the technique. With the feature extraction, we were able to get a training error of 0.2122 and a testing accuracy of around 0.60895 on the public data, which is better than what we had before.

Then, we tried dropping some of the less-important features, such as $ask2-5$, $bid2-5$, $ask2-5vol$, and $bid2-5vol$ while keeping our extracted ones. From this, we got a training error of 0.2123 and a testing accuracy of around 0.60887, so this was actually worse than before. Thus, we decided to keep the model with all of the data, including the newly extracted features.

Details of Models and Techniques

We considered using ensemble methods such as boosting decision trees (i.e., sklearn's gradient boosting, ada boosting, and cat boosting). Our gradient boost regressor from scikit-learn with 500 estimators, a max depth of 4, a minimum sample split of 2, and a learning rate of 0.01 had a training error of 0.2145. The AdaBoost regressor with 500 estimators, a learning rate of 0.01, and squared loss had a training error of 0.2183. Thus, we decided to stick with CatBoost's gradient regressor of a training error of 0.2150 since even though sklearn's gradient boost had a slightly better error than CatBoost's gradient regressor, CatBoost trained significantly faster, so we prioritized this time improvement over the slight error decrease. We also decided to try CatBoost's gradient classifier with the same parameters and log loss, and used the predict probabilities function to predict the probability that the data points were in class 1; that is, the probability that the mid-price will increase. This ended up giving us even better accuracy than CatBoost's gradient regressor, as we got a testing accuracy of 0.61210. Thus, we decided to use both CatBoost's gradient regressor and CatBoost's gradient classifier to test what hyperparameters, scoring, and validation models work best.

For each hyperparameter, we chose a set of candidate values based on our research and trained models that only differed by the hyperparameter that we focused on. For learning rates, we considered values of $\{0.01, 0.05, 0.075, 0.1\}$ and found that a learning rate of 0.1 gave the best error of 0.209. For regularization strength, we considered values of $\{1, 2, 3, 4, 5\}$ and found that a regularization strength of 5 gave the best error of 0.212. For maximum depth, we considered values of $\{1, 4, 7, 10, 15\}$ and found that a max depth of 15 gave the best error of 0.18. Finally, for the number of estimators in the random forest, we considered values of $\{250, 500, 750, 1000, 1250\}$ and found that 1250 estimators gave the best loss of 0.211.

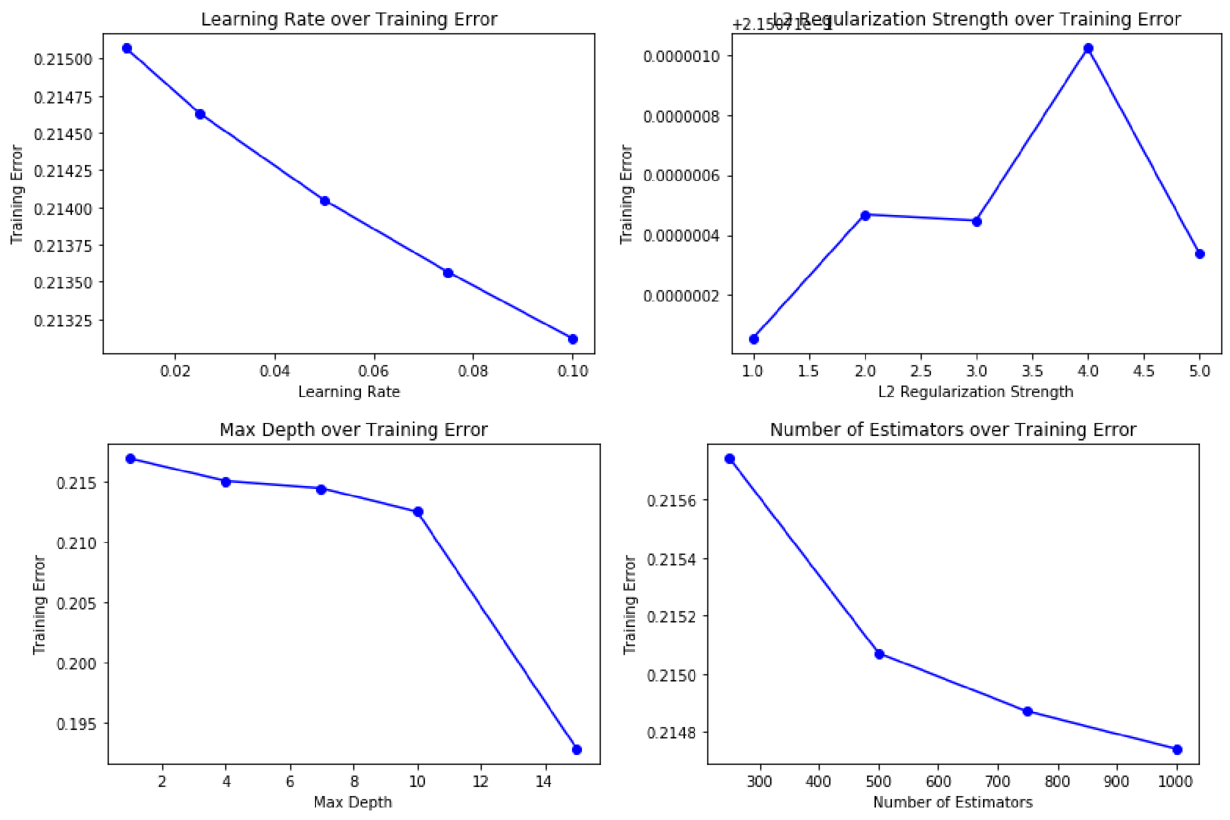


Figure 1: **Top Left:** A graph of training error vs. learning rate plotted using values of 0.01, 0.05, 0.075, and 0.1. **Top Right:** A graph of training error vs. regularization strength plotted using values of 1, 2, 3, 4, 5. **Bottom Left:** A graph of training error vs. max depth plotted using values of 1, 4, 7, 10, 15. **Bottom Right:** A graph of training error vs. number of estimators using values of 250, 500, 750, 1000.

However, using hyperparameters that decreased the training error too much gave us overfitting, since using the best values gave us a testing accuracy of 0.59477, which is lower than what we had got with standard parameters of 500 estimators and a maximum depth size of 4. Thus, we decided to use parameter values that gave us better testing accuracy at the cost of training error. These values turned out to be a learning rate of 0.1, regularization strength of 5, maximum depth of 10, and number of estimators 1000, giving us a testing accuracy of 0.61217.

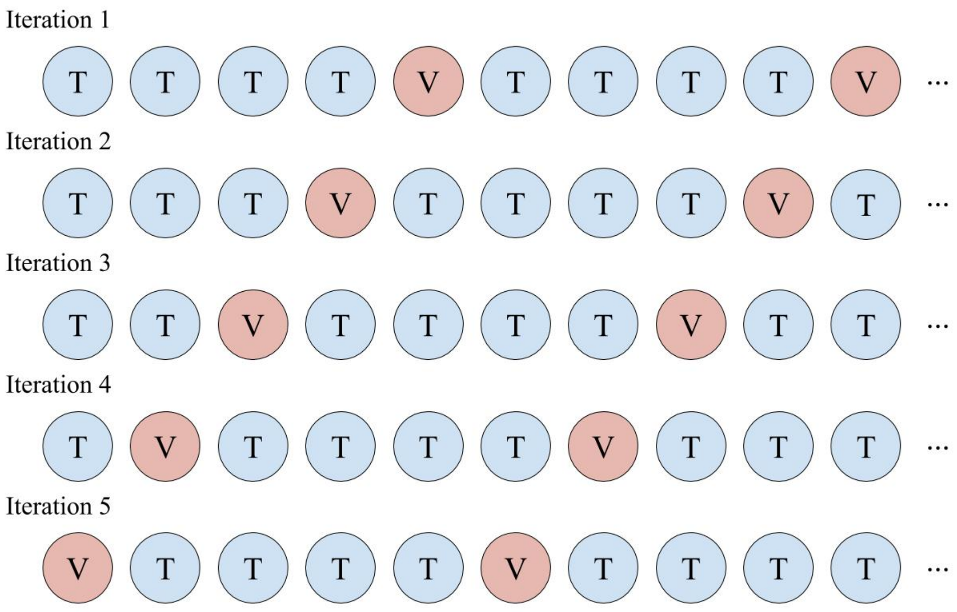
4 Model Selection [20 points]

Scoring

The optimization objective that we used is log loss, since the AUC function is not differentiable and thus cannot be optimized directly using gradient descent. We used log loss as it is suitable for a binary classification problem where we want to find the probability of a certain classification. We first tried using the built-in score functions for CatBoost, but the metric that CatBoost used to score was very different scale-wise from the metric than a mean-squared-error would use to score. Thus, we used scikit's mean squared error function to calculate the training error of the various models. As a standard, the default CatBoost gradient regressor gave a training error of 0.2150 while with cross-validation CatBoost gradient regressor gave a training error of 0.102. However, this did not necessarily indicate a better model, since the default CatBoost gradient regressor gave a testing accuracy of 0.59385 while the cross-validation CatBoost gradient regressor gave a testing accuracy of 0.59477. This could mean that our model with cross-validation was overfitting, resulting in a not-much-improved testing accuracy. Thus, we decided to try lowering the amount of estimators we were using.

Validation and test

We used cross-validation with 5 folds such that 80% of the training set was used for training and 20% was used for validation. Because the training set is chronologically ordered, We made sure to interlace the data points used for training and validation. If we split the data completely randomly, there could be an uneven distribution of points in similar time periods as opposed to points chosen evenly across the time dimension. Thus, picking every 5th point to be in the validation set and shifting the points picked by a position in order to get the validation set from a different fold yields much better results than completely randomized k-fold splitting.



We tested our model using the validation sets created by the splitting technique described above. Using cross-validation and feature extraction, we got a testing accuracy of 0.60112, which was actually lower than our original testing accuracy of 0.61217. Testing the model again using cross-validation but dropping some features as $ask2-5$, $bid2-5$, $ask2-5vol$, and $bid2-5vol$ while keeping our extracted ones gave us a testing accuracy of 0.60060, which was still less of our original testing accuracy of 0.61217. Thus, we decided to keep our original model without cross-validation.

5 Conclusion [20 points]

Insights

- The feature with the most influence on the prediction is the ask1vol feature. This could be because the quantity of contracts in the order book at the 1st ask price (the best/lowest/cheapest one) could determine how likely the mid price was going to behave well, since more contracts at the best price could indicate the mid price going down. We got the following top ten most influential features:

  1. ask1vol - positively influence the prediction target
  2. ask5 - negatively influence the prediction target, since the actual value of the 5th lowest ask price should not affect the mid price too much
  3. ask2vol - positively influence the prediction target
  4. bid1vol - positively influence the prediction target
  5. bid4vol - positively influence the prediction target
  6. ask5vol - positively influence the prediction target
  7. bid3vol - positively influence the prediction target
  8. ask3vol - positively influence the prediction target
  9. bid2vol - positively influence the prediction target
  10. ask4vol - positively influence the prediction target

  We found that our extracted features, while they had more influence than the ask and bid features, did not have as much influence as the volume features. This could be because the price data contained more extracted features (such as spread), leading to less influence by individual features.

- AUC is used because for two data points p_1 and p_0, where p_1 is classified as one and p_0 is zero, AUC measures the probability that the classification of p_1 is strictly greater than that of p_0. In this case, we are measuring the probability that the mid price will strictly increase. Thus, AUC lets us compare probabilities that range from 0 and 1 instead of just boolean values (ie. 1 for yes and 0 for no).

- Our method of validation could be parallelized since we not only test multiple hyperparameters, but for each candidate value for the hyperparameter, we train different models. Thus, parallelization would have allowed us to run these different models at the same time and compare results much more quickly.

- We learned that a Neural Net (NN) is not always required for data prediction, as we found that gradient boosting was sufficient for our data. We also learned that just because we have a low training error, the testing accuracy can be much worse, since real data is not necessarily entirely predictable, and since there is a higher chance that we are overfitting to our training data.

- Overall, we learned that while choosing a model is important, other steps of the process–including preprocessing data, choosing the key features to train on, and optimizing hyperparameters–are crucial to leveraging the greatest capabilities of our model.

Challenges

- CatBoost Regressor gave negative values and values of over 1, which could be because the regressor does not inherently measure probability. Thus, we opted for the CatBoost Classifier and utilized its built in probability function that calculated the probability that a given point's classification is 1.
- Our models started overfitting to the training data, since we found that our best hyperparameters and cross-validation attempts gave us lower training error but also lower testing accuracy. Thus, we tried to use hyperparameters, like higher regularization strengths, that gave lower training accuracy but higher testing accuracy to account for the overfitting.
- Our models took a very long time (4-5 hours with a GPU and a TPU) to finish training with our best hyperparameters. This is likely due to the high number of estimators, large max depth size, and increased number of models to train with cross-validation. Thus, we opted for a lower amount of estimators and smaller max depth size to give us results quicker and also models that did not overfit to the training data as much. These models took 2-3 hours to run in comparison.
- We could have tried testing our hyperparameters without cross-validation instead of with, since this testing took much longer than expected to run (2-3 hours). If we could redo this process, we would test different hyperparameters and models without cross-validation and all our extra feature extraction instead of trying to compare models with an almost-complete process, since this takes a long time. One alternative is implementing automatic hyperparmeter tuning using a sklearn package, for instance, which we unfortunately did not consider for this project.