Group name: group

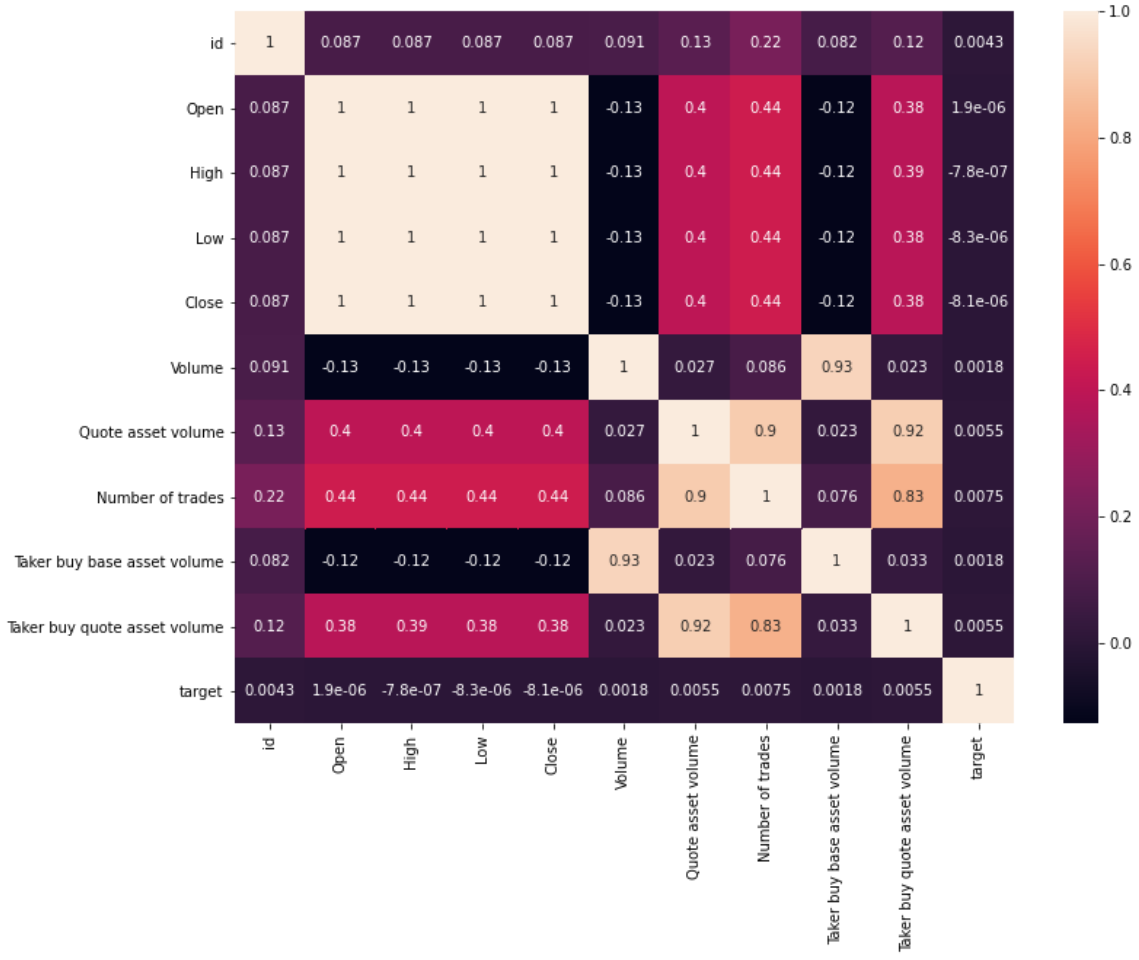
Fan Vincent Ka Chun (27754774)
Wong Chin Hang (20123808)

Exploratory Data Analysis

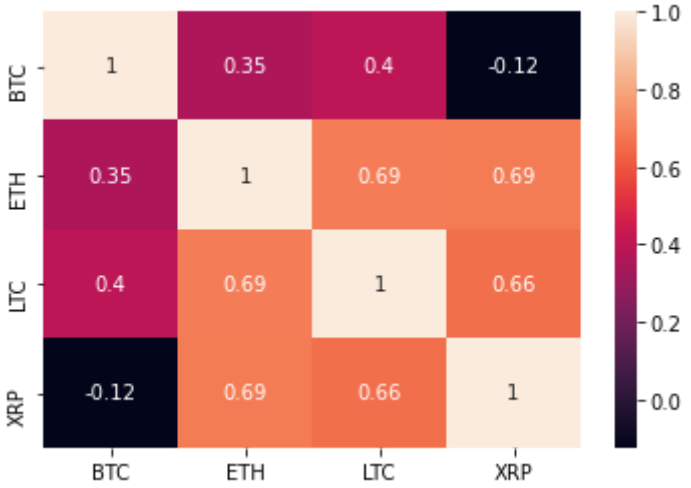
Before we began to create machine learning models to solve the prediction problem, we first explored and performed feature engineering on our dataset.

The first step is to check the completeness of the dataset and perform any data cleansing necessary. Since the dataset contains four different time series each corresponding to each currency, we first grouped the dataset by time and filter all dates that have entries different from four. We found that both the training data and testing data are not clean. For the training data, there are some duplicated rows, so we dropped all duplicated rows in the training data. The testing data also contains some duplicated rows, however we decide not to remove those rows for tree base models because the tree base model can handle them. On the other hand, we decided to drop them for the VAR model.

Then we took a closer inspection at the data distribution. First, we explored the correlation between each given feature which we computed the correlation matrix shown below. We found that prices columns, namely Open, High, Low, and Close, have a correlation very close to one. Whereas the volume columns, namely Volume, Quote asset volume, Taker buy base asset volume and Taker buy quote asset volume, have a high positive correlation. We also explored the correlation among cryptocurrencies.



From the correlation matrix shown below, none of the cryptocurrencies have very strong correlation. However, we know that there are some degrees of dependencies among them.



We also conducted the Augmented Dickey Fuller Test to examine stationary of the features. The Augmented Dickey Fuller Test is a unit root test for stationarity. Many time series models require the time series to be stationary. If a time series exhibits unit roots, the time series is a stochastic trend. Hence the time series is not stationary and may cause unpredictable results. The null hypothesis of the Augmented Dickey Fuller Test is that there is a unit root in the time series data. So, we want the p-value to be small to reject the null hypothesis and accept the alternate hypothesis that the time series is stationary.

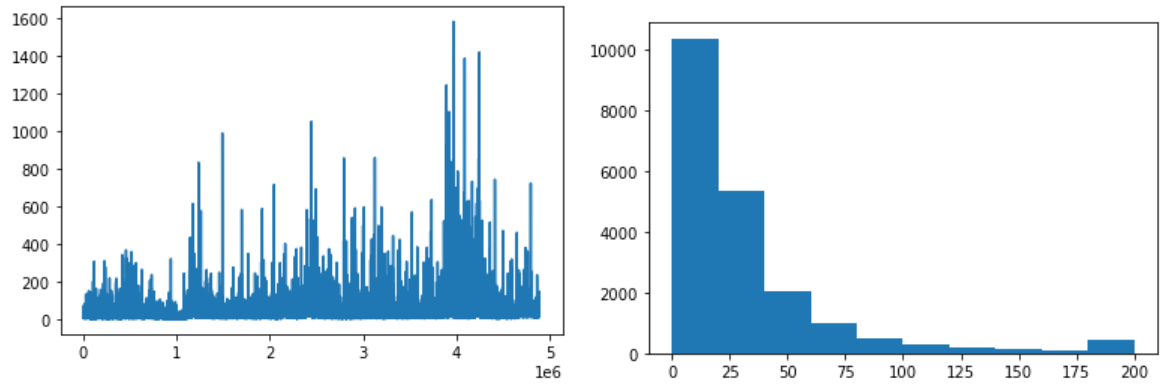
From the ADF tests, we found that the price of the time series is not stationary. For example, the opening price of BTC has ADF Statistic of -1.442851 and p-value of 0.561569. Since the p-value is large, we cannot reject the null hypothesis that there is a unit root. The time series is non stationary. However, if we use the log difference of the opening price instead of the price, the p-value now becomes 0.000000. Since the p-value is very small, we can reject the null hypothesis that there is a unit root. Thus, the time series is stationary.

For the Augmented Dickey Fuller Test of other features like volume and number of trade, we reject the null hypothesis as the p value is very close to zero. After data preprocessing, we apply log transformation on these features. The Augmented Dickey Fuller Test shows that the log transformation preserves stationary.

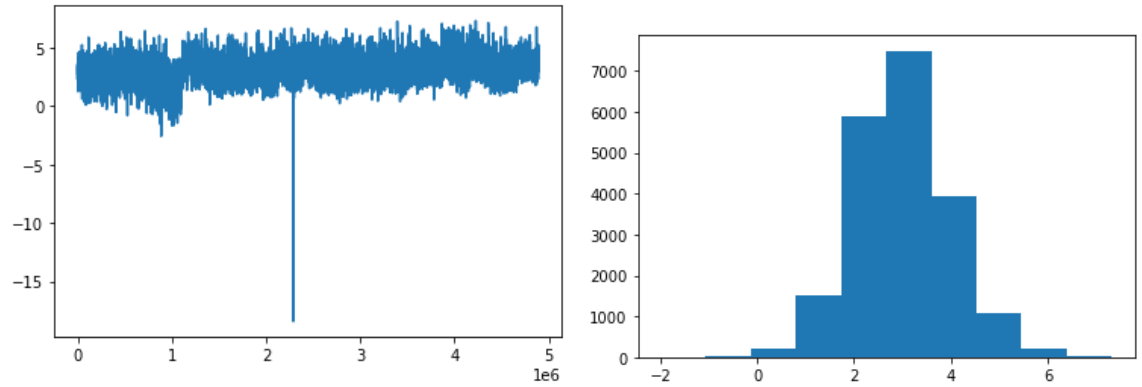
Data preprocessing

After exploring the dataset, we amended some of the features to facilitate modeling.

Firstly, we applied log transformation on the volume columns mentioned above. The following graphs show the distribution of BTC trading volume. The histogram of BTC trading volume clearly showed that the volume is not following normal distribution.



The graphs below show the distribution of BTC log trading volume. The histogram of BTC log trading volume showed that the volume is close to normal distribution.



Secondly, we take the log differences on the prices columns mentioned above. The reason is that the log difference of price is approximately the percentage change of the price if the log difference is small. Since our dataset's time interval is one minute and log difference is very small, log difference of the prices is a good approximation of percentage change of the prices. There are some merits of using percentage change of the prices instead of change of the prices. In general, change of price has positive correlation with price level. It means that the change of price has higher variance when the price level increases. On the other hand, percentage change of the prices is not correlated with the price level in general. Therefore, the variance is more likely to be the same than the variance of change of the prices across price levels.

Approaches

In this section, we will discuss the approaches we tried throughout the kaggle competition.

Vector autoregression

Vector autoregression is one of the most popular approaches to predict multivariate time series. The model is an extension of the univariate autoregressive model. Univariate autoregressive model is a forecasting model that is predicting future values based on regressing its past values. Vector autoregression allows regressing on multiple time series data to capture their relationships.

Vector autoregression allows regressing on both endogenous and exogenous variables. Endogenous variables are variables that we will like to predict by the model, namely the target column in this project. While exogenous variables are variables that are considered given to facilitate prediction. In this project, there are features like price and volume.

Vector autoregressions are sensitive to the time ordering. Therefore, we specially handled the unclean testing dataset. We first drop all duplicated rows and then add them back after prediction. Their predicted value is the same as the predicted value of the duplicated entries. However, the test results we obtained from Kaggle were not promising.

Temporal Fusion Transformer

In this section, we will go through our attempt with TFT as well as difficulties we encountered, and finally why we did not use TFT at the end.

After listening to TA's advice, we began looking into Google Research's Temporal Fusion Transformer paper. The paper proposed a state of the art model for multi-horizon prediction problems. As our data and problem falls under the multi-horizon prediction problem, TFT is the perfect model for this competition. Moreover, since our data consists of 4 time series and we know that these time series are influencing each other given most crypto-currencies are influenced by BTC. In conclusion, we believed that TFT can capture the temporal relationships between the time series data.

Although the code was published along with the paper, we encountered difficulties with running the default code. However, we searched for other similar models on different libraries that hopefully have optimized its memory usage and hardware requirements. We tried the TFT model from the pytorch forecasting library which we were able to run the code with a smaller subset of our dataset, approximately 50k data. It crashed when we attempted to use the whole training set on both GPU and CPU training since both the GPU and RAM storage were not big enough to host the training set during runtime. With many different attempts of fixing it including decreasing batchsize and decreasing the model parameters, we ended up not being able to train the model at all. Therefore, we had to stop further developing on TFT and move onto our other approaches.

Gradient Boosting

Another popular approach to solving general forecasting problems is using gradient boosting regressors as it was shown to be very successful in the past Kaggle competitions despite its simplicity. In this section, we will go through our approaches to generating a good forecast with the given dataset using different gradient boosting libraries.

Before we dive into the different libraries, we must first understand why gradient boosting is good and why it may be good for this forecasting problem. Vanilla boosting is a process of building an ensemble of weak prediction models in sequential order, oftentimes decision trees are used as the weak learners due to their simplicity and low cost of implementation. Note that training or testing error of a model can be described as a sum of irreducible error,

bias, and variance according to the bias-variance decomposition. Whereas ensemble learning can effectively reduce bias as each subsequent weak learner is trained to reduce errors of previous learners. Therefore, gradient boosting can also effectively reduce the overall error, making it a good candidate for any task.

The difference between vanilla boosting and gradient boosting is that gradient boosting minimizes the overall loss by using gradient descent to decide when to add more weak learners, meaning that the weak learners are chosen such that it points in the most negative gradient direction. Therefore, this process is also known as iterative functional gradient descent algorithm. Note that different loss functions can be used as the objective function which offer a higher degree of flexibility as different datasets or objectives require different loss functions. In conclusion, the contribution of each weak learner is based on how each weak learner contributes to minimize the overall model's loss.

Despite knowing that gradient boosting can be very effective and generate precise prediction, we were not sure whether this approach will work well with our time series data since not many studies were done on applying gradient boosting for multivariate time series data. As mentioned above, gradient boosting builds upon a lot of weak learners. Therefore, we hypothesize that it is very likely for the gradient boosting model to not capture the correlation or dependency between the time series data as opposed to the temporal fusion transformer. Nevertheless, we decided to give it a try given the implementation and training time cost is very small.

XGBoost

XGBoost is arguably the most famous gradient boosting library as it offers GPU support and provides parallel tree boosting. Its principle is close to gradient boosting decision trees but with some differences.

XGBoost adds a regular term on the number of leaf and square of prediction score in each leaf to control the complexity of the model. This helps prevent overfitting and improves the generalization ability of the model.

XGBoost utilizes both first-order and second-order partial derivatives of the cost function, while GBDT only uses first-order derivatives of the cost function. Utilizing the second-order derivative results in faster and more accurate gradient descent. Another advantage is the model can perform leaf splitting without specific the form of the loss function.

Moreover, XGBoost supports multiple types of base classifiers, such as linear classifiers. The traditional GBDT uses all the data in each iteration, while XGBoost uses a strategy like that of the random forest and supports sampling of the data.
Traditional GBDT is not designed to deal with missing values, XGBoost can automatically learn the processing strategy for missing values.

LightGBM

Next, we tried using the Light Gradient Boosting Model (LightGBM) published by Microsoft which has a few noticeable differences to XGBoost. First of all, LightGBM uses Gradient-based One Side Sampling (GOSS) to reduce the overall training time. GOSS reduces the number of data points sampled for each weak learner based on the gradients of those data points. For example, those instances with small gradients typically mean that the model is already correctly classifying the target such that the training loss is small. Thus, we can say that these data points will not contribute to convergence as much as other data points that have gradients. In other words, if the model can already correctly classify them, move onto the ones that the model cannot.

On the other hand, LightGBM also uses Exclusive Feature Bundling (EFB) to effectively reduce the number of features which can reduce overall training time and memory consumption. The algorithm can reduce the feature space by bundling features that are almost mutually exclusive such that we can reduce the dimensionality and training time. The algorithm first constructs a graph with each feature being a vertex, it then builds edges with weight being the number of overlapping values between the vertices. Then, it will bundle vertices that have the highest degree and weights which correspond to very similar features. Therefore, EFB can reduce the feature space which means it can also reduce the overall training time and complexity of the data.

With these unique features, we believe that LightGBM can extract knowledge of the intertwined time series with EFB and GOSS which may lead to better performance than other gradient boosting libraries. In our experiment, we tried using 2 types of boosting type, gradient boosting and random forest.

CatBoost

Finally, we decided to try CatBoost which stands for gradient boosting with categorical features support. As the name suggests, the model puts its focus on the categorical features, which produced some promising results in previous Kaggle Competitions. The major difference that makes CatBoost stand out among the other boosting models is that it treats categorical features in the most optimized way. Its vector representation of categorical data consists of a combination of ordered boosting and response coding. This means that when CatBoost is building the weak learners, it only considers past data points which prevent leakage from future time points and cause target leakage during the categorical vectorization. Therefore, we believe CatBoost might be suitable for time series forecasting problems given its strong categorical learning.

On the other hand, it can also reduce feature space similar to LightGBM's EFB, except that it excels in dealing with categorical features. CatBoost combines multiple categorical features automatically given that the features are highly correlated and related. Then, CatBoost combines these features into a single feature in its feature space which leads to faster learning and higher computation efficiency.

With our data set, we can turn the temporal features into categorical features. For example, with 2018/09/01 10PM, we can translate this into 9th month, 1st day of the month, 22nd hour

of the day. Thus, we can create a lot of categorical features with our dataset. So, we believed that CatBoost is worth trying in our problem setting given its excellent categorical features support.

Final Model

After experimenting with each of the approaches mentioned in the previous section. None of the approaches generate satisfactory results with both training and validation RMSE. However, we noticed that some models have lower RMSE in some validation time frame. For example, LightGBM performs better in the first half of the validation set. Whereas CatBoost performs better in the second half. Therefore, we decided to come up with a way to combine these models. Thus, we build an ensemble on top of these gradient boosting models, also known as model stacking.

Model stacking is a type of ensemble where we gather multiple strong learners in a way such that we take the best part out of each model. In fact, most of the Kaggle Competitions winning teams said that they used model stacking consisting of multiple deep learning models. Therefore, a lot of evidence points towards that model stacking can improve the overall performance. In general, there are 2 ways to stack the models, in series and parallel.

Firstly, we can stack the models in a sequential order such that the output of the previous model is one of the inputs for the next model. This method treats each model as a layer in a much deeper network with the addition of skip connections.

Secondly, we can stack the models in a parallel fashion such that each model is trained independently. Then, the outputs from each model are combined into a single output. This method resembles the ensemble method where instead of weak learners, we combined the outputs of each machine learning model.

Since more research and competition results showed that the second method is more favourable, we decided to stack the model in a parallel fashion. Moreover, we tried to combine the final output in 2 different ways, simple average and an additional linear regression layer. For the simple average way, we simply add up all the predictions from each model, then we divide by the number of models. Of course, this did not give a good result since our goal of model stacking is to combine the different strengths of each model. Therefore, we decided to use a simple linear regression model from sklearn to fit the results on the validation set. We choose to fit on the validation set because we believe that the final test prediction should be closely related to the validation set.

For our training methodology, we used the first 95% data as the training data, and the last 5% as the validation data. Moreover, we used early stopping on all models to prevent overfitting. As for hyperparameter tuning, we mostly tuned on the number of nodes and maximum depths in each boosting model.

In conclusion, our final model consists of 2 LightGBM models, 1 CatBoost model, and 1 linear regression layer as the final layer.

Results

The results of each individual model is shown below:

	LightGBM gbdt	LightGBM rf	CatBoost
Train RMSE	0.003360	0.003329	0.003362
Valid RMSE	0.003151	0.003153	0.003151

On the other hand, our final stacked model archived RMSE of 0.003355 on our training set, and 0.003149 on our validation set. Our final model achieved the 4th rank in the Kaggle leaderboard with RMSE of 0.00422. Therefore, we can see a large gap between our training and validation RMSE against the testing RMSE. This indicates that our final model is not predicting properly, meaning that there is a lot of room for improvements.

Discussion

As we saw from the results, there is a large gap between our evaluation metric and the test results. Moreover, since we are not using VAR or TFT that are more suitable for multivariate time series regression tasks, we know that our ensemble will not perform as well as those models. This further proves that our hypothesis of gradient boosting models being able to capture the relationship between the time series was incorrect.

Apart from failing to detect the dependency of the time series, we believed that the model also lacks to extract knowledge from the temporal features that we created. For example, if we are prediction for today's target, the gradient boosting models will refer to all occurrences of data that resembles today's temporal features. Such as the same hour of the day, same day of the month, or same month of the year. However, this did not work well as we know that crypto currencies prices increased abnormally after our training data. Thus, there was no seasonality to their prices and returns. In conclusion, since we saw that the time series have little to no seasonal cycles, and the gradient boosting models can only refer to previous similar encountered temporal features, our final model will undoubtedly perform worse in the further predictions.

To conclude our project, we tried many different approaches over the semester. Our finalized model was a stacked gradient boosting model consisting of LightGBM and CatBoost models. However, the final results were shown to be lacking due to lack of capability of capturing temporal relationships between the time series by the gradient boosting models.

Reference

Statsmodels VAR's Documentation. Statsmodels-developers. (n.d.). https://www.statsmodels.org/stable/generated/statsmodels.tsa.vector_ar.var_model.VAR.html

Lim, B., Arik, S. O., Loeff, N., & Pfister, T. (2020, September 27). *Temporal Fusion Transformers for Interpretable Multi-horizon Time Series Forecasting*. arXiv.org. https://arxiv.org/abs/1912.09363.

Grosse, R., Farahmand, A.-massoud, & Carrasquilla, J. (n.d.). *CSC 411 Lecture 5: Ensembles II*. https://www.cs.toronto.edu/~rgrosse/courses/csc411_f18/slides/lec05-slides.pdf.

CatBoost's Documentation. CatBoost. (n.d.). https://catboost.ai/.

LightGBM's Documentation. Welcome to LightGBM's documentation! - LightGBM 3.2.1.99 documentation. (n.d.). https://lightgbm.readthedocs.io/en/latest/.

Wolpert, D. H. (2005, October 18). *Stacked generalization*. Neural Networks. https://www.sciencedirect.com/science/article/abs/pii/S0893608005800231.

Xgboost's Documentation. Xgboost developers. (n.d.). https://xgboost.readthedocs.io/