Bitcoin Futures Forecast

Guowen Liu
Trinity Collage Dublin
Machine Learning
liugu@tcd.ie

Xiang Mao
Trinity Collage Dublin
Machine Learning
maoxi@tcd.ie

Xiangyu Zheng
Trinity Collage Dublin
Machine Learning
xizheng@tcd

1. INTRODUCTION

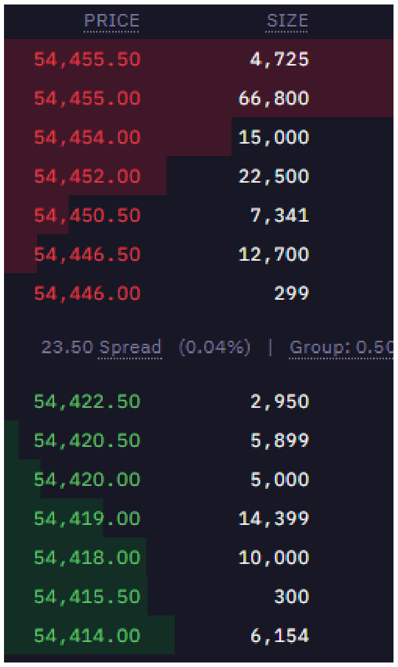
In this project, we collect two batches of Bitcoin Futures prices from Kraken Futures [1] in different periods, and use three machine learning models to predict the future price after 10 seconds.

2. DATASET AND FEATURES

2.1. Order Book

To collect the latest Bitcoin Futures prices, we developed two clients using Kraken Futures's API , written in C++ and Go respectively. They used WebSocket [2] protocol to subscribe to price updates sent by Kraken Futures in real time. For each client, we conducted performance tests, especially for network latency. In most cases, the C++ client showed lower latency, so we chose it as our data collector.

The prices at a given moment are organized into an order book, as shown in Fig. 1. It consists of two ladders: an ask ladder (*red*) and a bid ladder (*green*). For example, there are 299 ask orders priced at \$54446.00 and 2950 bid orders priced at \$54422.50. If they are consumed, the following behind them will move to their place. All our features and target values came from this order book.



PRICE	SIZE
54,455.50	4,725
54,455.00	66,800
54,454.00	15,000
54,452.00	22,500
54,450.50	7,341
54,446.50	12,700
54,446.00	299
23.50 Spread (0.04%) | Group: 0.5	
54,422.50	2,950
54,420.50	5,899
54,420.00	5,000
54,419.00	14,399
54,418.00	10,000
54,415.50	300
54,414.00	6,154

Figure 1. An order book consisting of two ladders

2.2. Hand-designed Features

The Bitcoin Futures price is different from normal buy and sell market values. It is based on supply and demand markets. We used 10 sets of data to construct the current real price. The steps are as follows:

1) We defined the middle price of *Size* N is the middle of an ask price and a bid price. At these ask and bid prices, the cumulative numbers of ask and bid orders reach N respectively. For example, the middle price of size 10000 is
$$\frac{54446.50 + 54420.00}{2}$$
in Fig. 1. The 54446.50 is the second price of the ask ladder (from middle to top). The cumulative number of ask orders $(299 + 12700)$ reaches 10000 at this price. The same principle applies, the 54420.00 is the third price of the bid ladder (from middle to bottom), at which the cumulative number of bid orders $(2950 + 5899 + 5000)$ reaches 10000.
2) We defined the middle price of *Level* N is the middle of the Nth ask price and the Nth bid price. For instance, the middle price of level 2 is
$$\frac{54446.50 + 54420.50}{2}$$
in Fig. 1. \$54446.50 is the second ask price and \$54420.50 is the second bid price.
3) We defined the *Benchmark Price* at a given time as the average of the following middle prices at that time:
   - Size 1000
   - Size 3000
   - Size 10000

- Size 30000
- Size 100000
- Level 1
- Level 3
- Level 5
- Level 8

4) With collected prices, calculated middle prices and benchmarks, we can create a middle price data frame as shown in Table. 1, where i means time. In order to use it to train models predicting the future benchmark after 10 seconds, we shifted the X_{26} column (benchmark) upward by 10 seconds and make it become y, as shown in Table. 2, which is the final data frame for model training.

3. METHODS

We use three algorithms to solve this prediction problem: Lasso Regression, K-Nearest Neighbors and Support Vector Regression with Gaussian kernel.

Lasso Regression finds the best weight for each feature by maximizing the *Negative Mean Absolute Error*. The cost function is shown as Equation. 1. The main reason for using Lasso Regression is that it can reduce the weight of some features to zero. It can help us to select the most important data from the manually designed features. We believe that many features are redundant.

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}|y_i - \widehat{y_i}| + c\sum_i |\theta_i| \tag{1}$$

K-Nearest Neighbors is a non-parametric method. It makes predictions by comparing an unseen sample to the nearest k number of known samples. The output is the average of the values of these k neighbors. With this model, we wanted to test if similar ladders would lead to similar future prices. Another crucial reason we used it is that a K-Nearest Neighbors model does not need to be retrained after training data updated, which allows it to be easily integrated into the client, circularly receiving price updates from the server, predicting future prices and sending orders to the server.

Support Vector Regression is different from the traditional regression model. As shown in Fig. 2, the area formed by the two dashed lines is called the interval zone. If the sample falls into this area, it is considered that the prediction is correct and the loss is not calculated. The size of the interval zone is determined by the distance from the sample to the hyperplane. Our goal is to find the maximum interval to divide the hyperplane.

We used Gaussian kernel (Equation. 2) to map data to high dimensions, making the originally linearly inseparable data separable.

$$k(x_i, x_j) = exp\left(-\frac{||x_i - x_j||^2}{2\alpha^2}\right) \tag{2}$$

1. https://en.wikipedia.org/wiki/Support-vector_machine

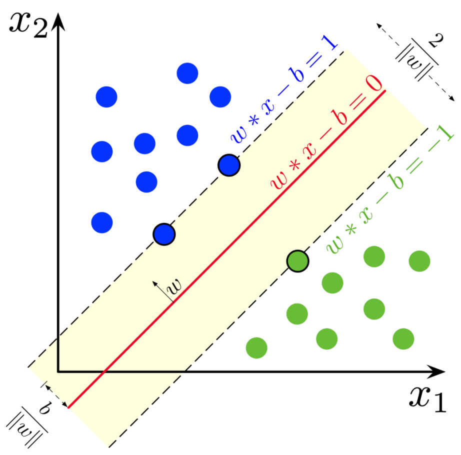


Figure 2. Support Vector Regression[1]

4. EXPERIMENTS

All experiments were performed with 5-fold cross-validation, implemented by `cross_val_score` function in `scikit-learn` library. The returned scores or errors are evaluated on test folds.

4.1. Baseline Regression

We set a baseline of the mean absolute error across all features for prediction, and the mean absolute error for 5-fold cross-validation was \$203.06 with a standard deviation of \$106.27.

4.2. Lasso Regression

Although we intuitively thought that polynomial features cannot improve the accuracy of prediction, we still did tests for it using Linear Regression without regularization, from degree 1 to 4. The mean and standard deviation of absolute errors in 5-fold cross-validation are shown in Fig. 3. With the degree increasing, the model becomes over-fitted and unstable. So we abandoned polynomial features.

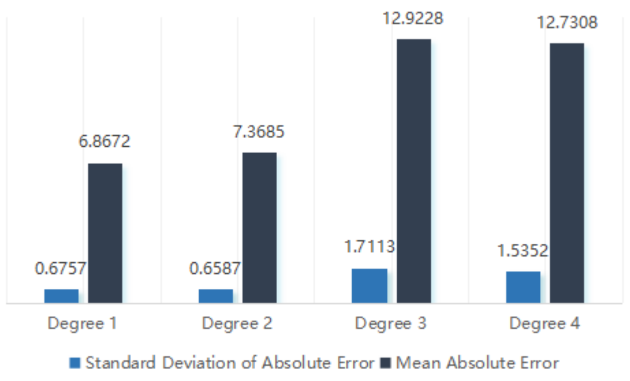


Figure 3. Mean absolute errors with different degrees

TABLE 1. THE COLLECTED MIDDLE PRICE TABLE

X_1	X_2	X_3	X_4	X_5	X_6	$\cdots$	X_{25}	X_{26}
Size 1000_i	Size 3000_i	Size 10000_i	Size 30000_i	Size 100000_i	Level 1_i	$\cdots$	Level 20_i	Benchmark price_i
Size 1000_{i+1}	Size 3000_{i+1}	Size 10000_{i+1}	Size 30000_{i+1}	Size 100000_{i+1}	Level 1_{i+1}	$\cdots$	Level 20_{i+1}	Benchmark price_{i+1}
Size 1000_{i+2}	Size 3000_{i+2}	Size 10000_{i+2}	Size 30000_{i+2}	Size 100000_{i+2}	Level 1_{i+2}	$\cdots$	Level 20_{i+2}	Benchmark price_{i+2}
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$

TABLE 2. THE SHIFTED MIDDLE PRICE TABLE

X_1	X_2	X_3	X_4	X_5	X_6	$\cdots$	X_{25}	y
Size 1000_i	Size 3000_i	Size 10000_i	Size 30000_i	Size 100000_i	Level 1_i	$\cdots$	Level 20_i	Benchmark price_{i+10}
Size 1000_{i+1}	Size 3000_{i+1}	Size 10000_{i+1}	Size 30000_{i+1}	Size 100000_{i+1}	Level 1_{i+1}	$\cdots$	Level 20_{i+1}	Benchmark price_{i+11}
Size 1000_{i+2}	Size 3000_{i+2}	Size 10000_{i+2}	Size 30000_{i+2}	Size 100000_{i+2}	Level 1_{i+2}	$\cdots$	Level 20_{i+2}	Benchmark price_{i+12}
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$

After fixing the degree, we tested different penalty factor c of Lasso Regression, shown in Fig. 4. When c is 0.00001, the mean absolute error of 5-fold cross-validation is the lowest.

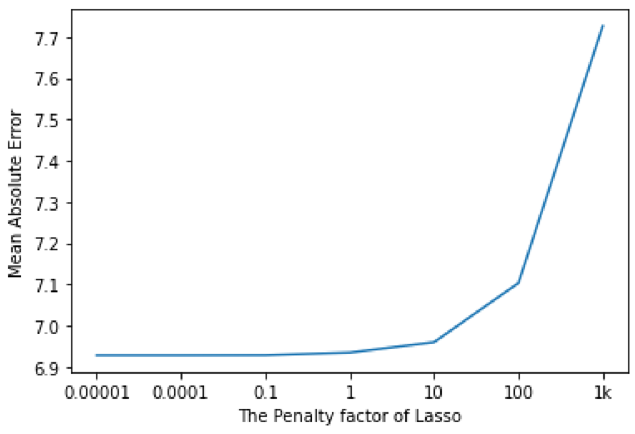


Figure 4. Mean absolute errors with different c in Lasso Regression

The top five non-zero weights for middle prices are shown in Table. 3. The weights of most of the remaining features are zero. With these hyper-parameters, the fitted model achieved a mean absolute error of \$6.94.

TABLE 3. THE NON-ZERO WEIGHTS FOR MIDDLE PRICES

Feature	Weight
Size 3000	0.9992995
Size 100000	0.0000470
Size 10000	0.0000406
Size 30000	0.0000318
Level 1	0.0000284

4.3. K-Nearest Neighbors

We tested different k values from 1 to 70, as shown in Fig. 5. It clear that when k is 30, the mean absolute error is the smallest, around \$7.15. If k is larger than 30, over-fitting causes higher errors.

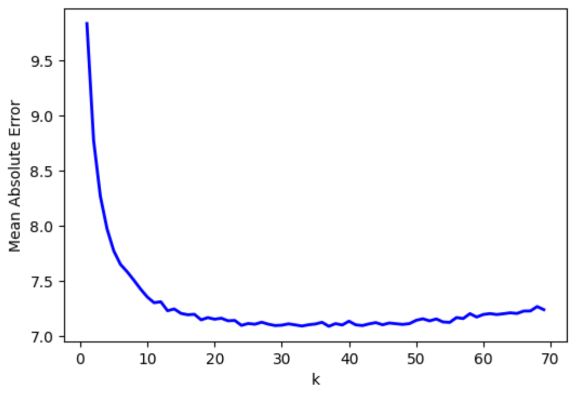


Figure 5. Mean absolute errors with different k in K-Nearest Neighbors

4.4. Support Vector Regression

In Fig. 6, we explored c values range 10 to 1000. Over-fitting and instability begin to appear when c is larger than around 300. It provided a mean absolute error of \$6.80.

5. SUMMARY

The lowest mean absolute errors achieved by each model are shown in Table. 4. In terms of model performance alone, Support Vector Regression with Gaussian kernel is slightly ahead. But the disadvantage is obvious. During the experiment, its running time is much longer than the other

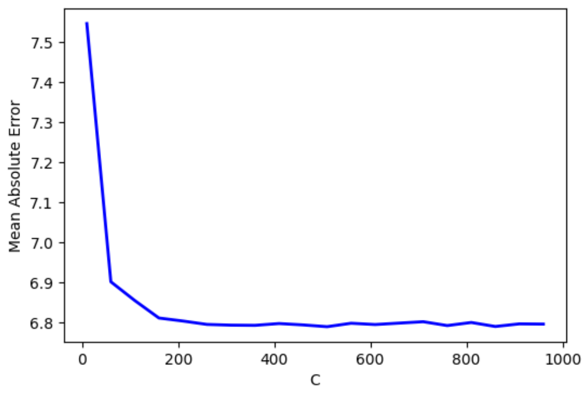


Figure 6. Mean absolute errors with different c in Support Vector Regression

two models. Although K-Nearest Neighbors is the worst, it can continuously run without being retrained. As mentioned before, we think it is an important characteristic for our application.

TABLE 4. MEAN ABSOLUTE ERROR OF EACH MODEL

Model	Mean Absolute Error
Baseline Regression	\$203.06
Lasso Regression	\$6.94
K-Nearest Neighbors	\$7.15
Support Vector Regression	\$6.80

We think that the following reasons have negative impacts on model accuracy:

- The accurate price of bitcoin cannot be predicted by market data alone. In addition to current market buying and selling conditions, external factors, such as national policy adjustments to bitcoin, can also have an impact on the price, represented as a sharp drop or rise without warning.
- The models we used are too simple and have no memory function. They cannot learn longer-term dependencies like Recurrent Neural Network and Long Short-Term Memory models.
- Time period of data collection. We collected two batches of data in different time periods. The first batch is relatively old and most benchmark prices range from \$32000 to \$33000, as shown in Fig. 7. The second batch shown in Fig. 8 is the latest but has more fluctuations. Most benchmark prices range from \$56000 to \$58000. The standard deviations of two batches are \$215.39 and \$499.89 respectively. The results mentioned earlier are based on the first dataset. We used the same hyper-parameters to train models on the second dataset, mean absolute errors were larger, ranging from \$10 to \$15.

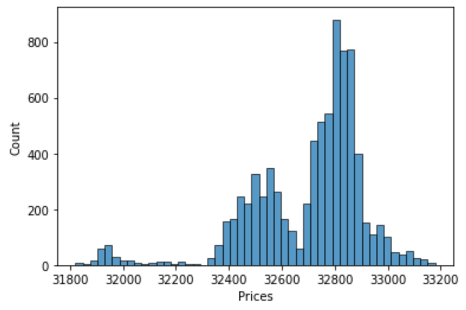


Figure 7. The old batch of Bitcoin Futures benchmark prices

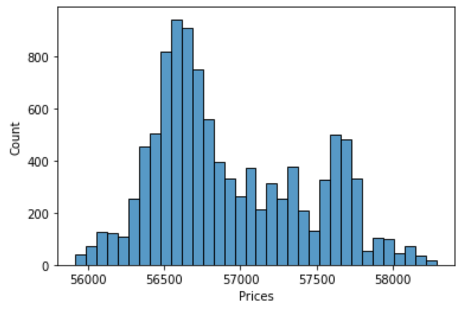


Figure 8. The new batch of Bitcoin Futures benchmark prices

6. CONTRIBUTIONS

Guowen Liu and Xiang Mao: Mainly responsible for code testing and development for data collection and linear regression. Also responsible for functional engineering. Participates in report writing for relevant sections. Github code is submitted and managed by Xiang Mao.

Xiangyu Zheng: Responsible for the testing and development of the SVR and KNN and for writing the relevant chapters in the report

7. CODE

GitHub: https://github.com/iczcpkqo/ml_group-project_66

References

[1] Kraken Futures, "Kraken Futures," Available at https://futures.kraken.com (2021/11/28), 2021.

[2] Wikipedia, "WebSocket," Available at https://en.wikipedia.org/wiki/WebSocket (2021/11/28), 2021.