Auto Regressive Integrated Moving Average Risk Management Strategy

Dimitri Chemla

June 2023

Contents

1 Introduction

Enigma's goal is to develop a toolkit of different statistical analysis tools, which can be used in conjunction, to generate real time, accurate market signals, for any market, in any condition. This model would serve along a few others we have already developed, namely a Bollinger Bands signal, as well an OrderBook Pressure indicator, which both show promising signs of accuracy and adaptability.
The newest addition is an AutoRegressive Integrated Moving Average (ARIMA), which is a form of regression analysis that determines the strength of a dependent variable relative to a changing variable. The purpose of the model is to predict future data points, whether it be price or others, by examining the differences between values in the series. In our case we are using Perpetual Bitcoin data (which follows spot data 1:1), to develop, test and optimise this model.

AutoRegressive (AR): refers to a model that shows a changing variable that regresses on its own lagged values.

Integrated (I): represents the difference of raw observations to allow the time series to become stationary. The purpose of making the data stationary is to render the data consistent and so remove trends and seasonality. Seasonality introduces predictability, which could negatively affect the predictions of this model

Moving Average (MA): incorporates the dependency between an observation and a residual error from a moving average model applied to lagged observations.

ARIMA(p,d,q) Parameters:

- p : number of lagged observations in the model
- d : the number of times raw observations are differenced
- q : size of MA window

There are a multitude of different ideologies regarding the configuration of these parameters, however for the sake of testing and development of the model, we have decided to use (4,0,1) as parameters for (p,d,q). Later, we will speak above optimisation of these parameters. The test/train split was at first set to 50-50, however after further thought, we realised that giving a majority of the data to the training of the model was more beneficial not only in terms of performance, but also simulates the real conditions of the model once in production, as it will have a constant inflow of data.

It is often said that giving a model too much data for a short term prediction,which is what we are trying to achieve, may be counterproductive, as there may be either too much noise for the model to give an accurate prediction or there could also be computational issues, wherein the model would take longer to run and give predictions than how far in the future it is trying to predict, or maybe even issues of over-fitting. To overcome any of these issues, the totality of the data fed to the model equates to around 4-5 weeks of OHLCV minute data.

It is worth exploring how the model performs with different amounts of train and test data it has available, so that we can find the optimal amount to give it, as once the model is in production, it will have a constant inflow of new data and so can constantly be giving signals over the period of time we choose, more on this later.

How far into the future the model predicts could also be a source of optimisation, as this would correlate with the general accuracy of the signal, which is the KPI of our model. From the results section, we can see the difference in standard error as well as the % difference between predicted and actual values, and how this generally gets greater the further in the future we are predicting.(It is important to note that this not always the case, for example if there is a relatively extreme drop in price over a 1-day period, the model will be ahead of the dip but may miss out on potential smaller rises on the way down to the bottom.). We have seen that while giving the model around 5 weeks of data, prediction in the next half-day (900 minutes) was quite reasonable, however the model prediction starts to deteriorate around 3-5 days ahead.

Additionally, it would be important to quantify the success rate of the direction prediction over short/very short term predictions. The direction (whether the price moves up or down) would be an extremely important feature if we are trying to make predictions within the next 5m-5h.

2 Data

For the ARIMA model, the data used must be a univariate time series. In our case we are using 1 minute BTC Perpetual OHLCV data which spans from 01.01.2022 to 31.05.23, and take only the close data and the timestamp of the close price data, to build our univariate time series.
Once we have succesfully built and tested our initial model, there is a possibility to implement an adaptation of the ARIMA model, which enables us to include other variables, such as volatility, volume and others, converting our time series to a multivariate one, permitting us to use a model called VARIMA/X or SARIMA/X, which can take multiple factors into consideration when predicting future prices. The SARIMA/X model is more interesting as it is an adaptations of ARIMA which include seasonality and exogenous variables in the model. Cryptocurrencies are well known for their cyclical and seasonal movements and so including multiple variables as well as a seasonality factor would make sense, and hopefully lead to a more accurate prediction.
With this data we are trying to achieve very short/short term price prediction. Using this model to make predictions over a longer term would not be sensible as the model is quite untrustworthy over longer periods.
The main use of this model for us, is to generate an accurate and reliable directional signal rather than to try predict actual price values, (even though the model returns price points, we are more interested in the general direction of the price points rather than the actual prices themselves.). Of course it would be an added benefit that these predicted prices be as close to the price action as possible, which is why there is an average % error metric between predicted price and realised prices.

3 Implementation

Before we are able to implement the ARIMA method, we first need to process the raw data, which comes in the form of 500+ .csv files. The function **process_files()** will gather all information relating to *'PERP_BTC_USD'* from all .csv files and create a .txt file to store the gathered information. There then is a data processing function which will return the data in the correct format needed to apply ARIMA to it, which is a univariate time series of close prices.

Using the **statsmodels** library in Python, we are able to implement a built in function **ARIMA()**, which takes in 3 parameters, p,d,q and the univariate series. During the development of the model, these parameters were set to (4,0,1), however, after back-testing, we have found that there are other parameters which can yield higher overall accuracy.

```
def arima():
        #splitting the data into test/train sets
        size = int(len(df_close) * 0.90)
        train = df_close['px_close'][0:size]
        test = df_close['px_close'][size:len(df_close)]

        #applying ARIMA() method, with p,d,q paramters, obtained
    from a config file.
        model = ARIMA(train, order=(p,d,q))
        fitted = model.fit()

        #make predictions, n_pred into the future from the last
    known data point
        pred_result = fitted.get_forecast(n_pred, alpha=0.05).
    summary_frame()

        pred_idx = test[:n_pred].index
        pred_result.index = pred_idx
        #combination of historical and predicted values
        combined_df = pd.concat([df_close, pred_result])

        #metrics
        mean_vals = pred_result['mean'].values
        tst_vals = test[:n_pred]
        a_b = mean_vals-tst_vals
        prct = (a_b/tst_vals)*100
        avg_err = (sum(prct)/n_pred)
        mse = mean_squared_error(tst_vals, mean_vals)
        rmse = np.sqrt(mse)
        r2 = r2_score(tst_vals, mean_vals)
        bic = fitted.bic
        aic = fitted.aic

        metrics_dict = {'avg_err':avg_err, 'rmse':rmse, 'r2':r2, '
    BIC':bic, 'AIC':aic}
```

Listing 1: Application of ARIMA model

Below, we can see the direction() function which simply calculates the gradient between the last known value of the price series, and the gradient at the given intervals of interest, and then compares the truth value of the direction with the direction of the realised price, returning a final truth table of predicition direction vs realised direction, at each specific time interval.

```python
def direction(df_compar, actual, test):

    arra = [5,10,15,20,25,30,60,120,180,300,600,720,899]
    actu_strt = actual[0]
    pred_strt = df_compar['mean'][0]
    actu_grads = np.array([])
    pred_grads = np.array([])

    for a in arra:
        actu_val = actual[a]
        actu_grad = (actu_val-actu_strt)/(to_unix(test.index[a]) - to_unix(test.index[0]))
        if actu_grad > 0:
            actu_grad = 1
        elif actu_grad < 0:
            actu_grad = -1
        else:
            actu_grad = 0
        actu_grads = np.append(actu_grads, actu_grad)

    for a in arra:
        pred_val = df_compar['mean'][a]
        pred_grad = (pred_val-pred_strt)/(to_unix(df_compar.index[a]) - to_unix(df_compar.index[0]))
        if pred_grad > 0:
            pred_grad = 1
        elif pred_grad < 0:
            pred_grad = -1
        else:
            pred_grad = 0
        pred_grads = np.append(pred_grads, pred_grad)

    pred_ser = pd.Series(pred_grads)
    actu_ser = pd.Series(actu_grads)
    dir_df = pd.concat([actu_ser, pred_ser], axis=1)
    dir_df = dir_df.rename(columns={0:'actu_dir', 1:'pred_dir'})
    dir_idx = ['5m','10m','15m','20m','25m','30m','1h','2h','3h','5h','10h','12h','15h']
    dir_df.index = dir_idx
    comparison = dir_df['pred_dir'].values == dir_df['actu_dir'].values
    dir_df['dir_res'] = np.where(comparison, 1, 0)
    dir_df = dir_df.drop(columns=['actu_dir', 'pred_dir'])
    dir_df = dir_df.transpose()

    return dir_df, pred_result
```

Listing 2: Checking if prediction follows direction of actual values

4 Initial Results

ARIMA prediction over test period

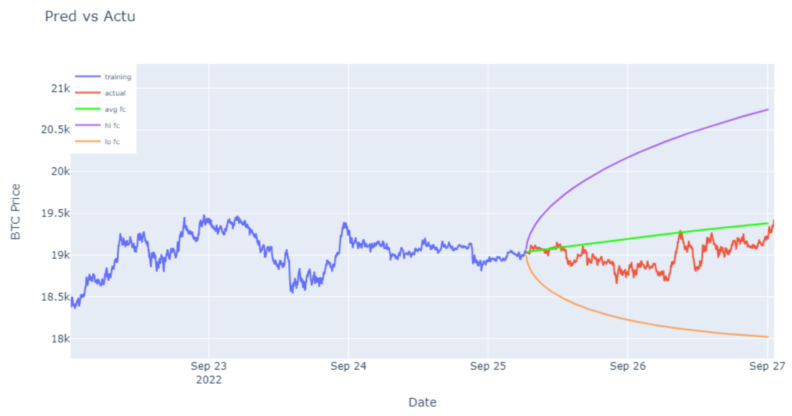


Figure 1: Over the test period, we can see that the general trend of the actuals is rather well predicted by the model.

Figure 1 shows the results of the prediction obtained from line 12 in the first code snippet, plotted against the timestamp of the time series, with the actual price action behind it in red. The ARIMA function returns a dataframe with upper and lower bounds of the prediction, as well as the average of these 2 bounds and the mean standard error. We are plotting the upper bound of the forecast (purple), lower bound of forecast (orange), the mean forecast (green) and the actual price of the test period (red).
As we can see, the model has correctly predicted the direction of the price action, and as an added bonus has hit key price points on the uptrend. This is a result obtained from ARIMA(4,0,1), so with no optimisation of the model, suggesting this may be a good combination for this specific market and market condition, however optimisation of the parameters is of course needed in order to ensure that we use the most best parameter combination, applicable to multiple market conditions.
As mentioned previously, the model has been able to predict the correct overall trend, but has not been able to account for the small dips before the uptrend, as so as we can see in Figure 2, the error % between the prediction and realised prices varies a little in the middle, before dramatically decreasing, one the uptrend is realised. This is one of the disadvantages of prediciting over longer periods, however through testing, we have seen that very short term predictions, predictions less then 5-10hrs in the future, are not only not accurate, but actually predict nothing, as the mean of the upper prediction and lower prediction will nearly always be flat.

Prediction % difference with actual price

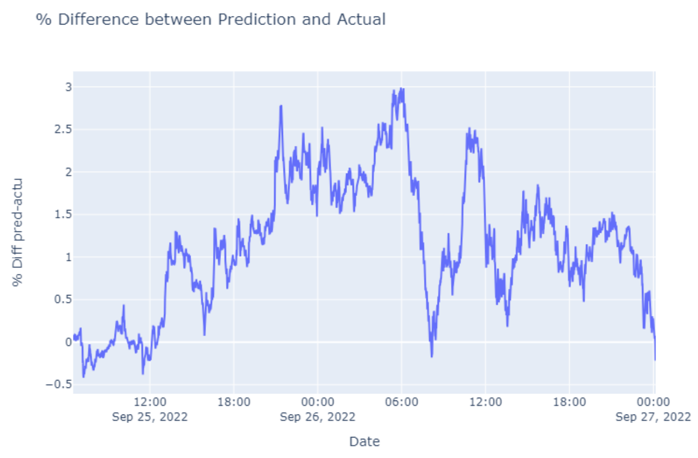


Figure 2: Shows clearly that the error is on average quite small, with an average error of 1.148% over the test period .

5 Optimisation & Results

For the optimisation of the ARIMA model, we thought it best to use the same method as in Bollinger Bands, where we used nested for loops to try a great amount of different hyper-parameter combinations, to see which combination/s yielded the highest PNL. In this application, our most important KPI is to know whether the direction of the signal is correct and for how long it is correct, ie. how far in the future is it accurately predicting.

```
n_pred = 900

    for j in np.arange(1, 630491, 1000):
        k = j + 50000
        for p in np.arange(0, 6, 1):
            for d in np.arange(0, 2, 1):
                for q in np.arange(0, 3, 1):

                    opti(p, d, q, j, k, n_pred)
```

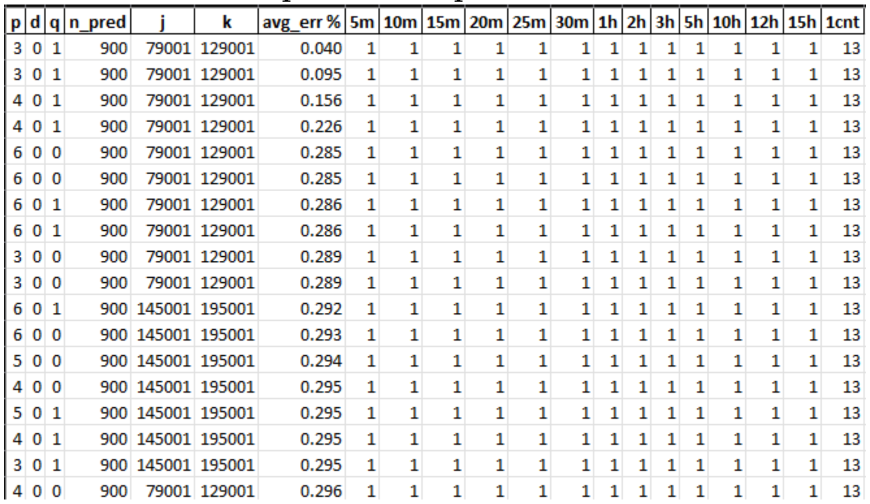
Listing 3: Running through all different types of parameter combination and compile KPI from each test

The code listing above shows a nested for loop which generates each combination of hyperparameter to be tested by the function opti().

For now we are using a single value for the number of predictions we are making, (n_pred), as we have found 900 to be of optimal range for testing purposes. To ensure a fair test and no overfitting of the model to the data, a variable **j** is randomised each time a new combination is ran, so that the model is tested on a different part of the 1.5yrs of BTC PERP data each time the loop is ran.

The j and k parameters are 50000 price points apart as this simulates a test window of around 5 weeks. From the back-test, the optimisation function returns a dataframe of many important features of the test, such as test parameters, test conditions and key metrics from the model test. The key part of the results is the directional indicator, which gives an insight on whether the test correctly predicted the direction over very short/short term, with directional signals ranging from 1 minute to 15 hours.

Top ARIMA parameters



p	d	q	n_pred	j	k	avg_err %	5m	10m	15m	20m	25m	30m	1h	2h	3h	5h	10h	12h	15h	1cnt
3	0	1	900	79001	129001	0.040	1	1	1	1	1	1	1	1	1	1	1	1	1	13
3	0	1	900	79001	129001	0.095	1	1	1	1	1	1	1	1	1	1	1	1	1	13
4	0	1	900	79001	129001	0.156	1	1	1	1	1	1	1	1	1	1	1	1	1	13
4	0	1	900	79001	129001	0.226	1	1	1	1	1	1	1	1	1	1	1	1	1	13
6	0	0	900	79001	129001	0.285	1	1	1	1	1	1	1	1	1	1	1	1	1	13
6	0	0	900	79001	129001	0.285	1	1	1	1	1	1	1	1	1	1	1	1	1	13
6	0	1	900	79001	129001	0.286	1	1	1	1	1	1	1	1	1	1	1	1	1	13
6	0	1	900	79001	129001	0.286	1	1	1	1	1	1	1	1	1	1	1	1	1	13
3	0	0	900	79001	129001	0.289	1	1	1	1	1	1	1	1	1	1	1	1	1	13
3	0	0	900	79001	129001	0.289	1	1	1	1	1	1	1	1	1	1	1	1	1	13
6	0	1	900	145001	195001	0.292	1	1	1	1	1	1	1	1	1	1	1	1	1	13
6	0	0	900	145001	195001	0.293	1	1	1	1	1	1	1	1	1	1	1	1	1	13
5	0	0	900	145001	195001	0.294	1	1	1	1	1	1	1	1	1	1	1	1	1	13
4	0	0	900	145001	195001	0.295	1	1	1	1	1	1	1	1	1	1	1	1	1	13
5	0	1	900	145001	195001	0.295	1	1	1	1	1	1	1	1	1	1	1	1	1	13
4	0	1	900	145001	195001	0.295	1	1	1	1	1	1	1	1	1	1	1	1	1	13
3	0	1	900	145001	195001	0.295	1	1	1	1	1	1	1	1	1	1	1	1	1	13
4	0	0	900	79001	129001	0.296	1	1	1	1	1	1	1	1	1	1	1	1	1	13

Figure 3: Truth table to see if direction of prediction follows direction of realised prices for given time period after last known value.

Above we can some of the best results for the back-testing of the ARIMA model. Here are the top 20 results of over 5000 combination tests. Of all the tests, there are 279 test cases of the 5000 that have 100% hit ratio. In total, around 50% of results have a hit ratio of above 75%.

From these results, we are able to determine that the combinations (3,0,1), (4,0,1), (5,0,1), (6,0,1) were the most ideal to cover basically any market condition that was present over the 1.5 year test period.

6 Improvements & Next Steps

The next steps are to combine the ARIMA, Bollinger Bands and OBP indicators into one big signal which uses the ARIMA model to predict future price action directions and to program a logic which uses the BB and the OBP in order to confirm the prediction, and send a final market entry/exit signal. The idea is that the OBP will give the final confirmation, as this is the signal which takes into account the market microstructure, and so should be the most deterministic for price action. The BB will also act as a confirmation to ensure that the overall trend is backed by other stats.