SC1 Group Project - Lab Notebook

Henry Bourne, Sam Perren, Dylan Dijk

2023-01-13

Contents

Forecasting Bitcoin Price Using Support Vector Machines

Our project looks at trying to create a short term forecasting model for Bitcoin price in USD (United States Dollar). We can roughly divide our process of trying to create such a forecasting model into three sections:

1. Selecting predictor variables.

2. Fit a support vector regression (SVR) model using our predictor variables.

3. Validate the performance of our regression model (against other models we've fitted).

In this report we describe how we carried out each section, and provide the code alongside that produced our results.

First round of analysis

We will start by producing a model and some graphs that will allow us to see the model we've fitted, in the next section we will then create some other models which will allow for a comparison between models in the final section.

Choosing predictor variables

The first thing we must do before fitting a regression model is choose our predictor variables, in certain scenarios it can be fairly straight forward picking what data may be instrumental in creating a good model of your target variable. However, financial data is notoriously difficult to create models for, in part because it's not clear what exactly drives the price of any given stock, commodity or currency. Most probably its a very diverse set of variables and also probable that selection of variables and their relative importance also evolves over time. Here we will be investigating creating a model for Bitcoin price based on the price action of other assets traded on exchanges, the idea here being that price action in other assets may signal what will happen to Bitcoin price.

Now the question here is how do we select which exchange traded assets to include as predictor variables in our model? as discussed in the SM1 report one way of doing this is to use the `glasso` package to select the predictor variables that are the most instrumental in creating a model for Bitcoin price.

We can extract the stock tickers for the S&P 500 using `tq_index()`. The stocks are then listed in descending order with respect to their weighting in the index. We can then use this to import the stock data for each of the stocks in the S&P 500. In addition to the S&P 500 we will look include price data from other crypto currencies, stocks of companies related to the crypto-currency space and some large indices related to precious metals, energy and commodities.

Loading S&P 500 stock data and additional Cryptocurrencies

```
SP500 <- tidyquant::tq_index("SP500")
```

```
> Getting holdings for SP500
```

```
SP500_symbols <- SP500$symbol
tickers <- unique(c("BTC-USD", "ETH-USD", "BNB-USD", "USDT-USD",
    "ADA-USD", "DOGE-USD", "XRP-USD", "LTC-USD", "TSLA",
    "NVDA", "AMD", "PYPL", SP500_symbols))
data_full <- tidyquant::tq_get(tickers, from = "2019-01-01")
```

Formatting the dataset.

```
data <- data_full[, c("symbol", "date", "adjusted")]
data <- data %>%
    tidyr::pivot_wider(names_from = symbol, values_from = adjusted)
data <- data[, -1]
```

Using `glasso()` to select predictors

Now we have imported all the data we need we can calculate the covariance matrix accounting for missing data.

```
data <- as.matrix(data)
covs = cov(data, use = "pairwise.complete.obs")
```

Now we can use the `glasso()` function from the `glasso` package to use lasso techniques to estimate a sparse covariance matrix, we can plot the resulting precision matrix:

```
g_results <- glasso::glasso(covs, rho = 3e+06)
```

Let's now look at which stocks had a non-zero covariance with Bitcoin:

```
btc_g_results <- tickers[which(g_results$w != 0)]
btc_g_results <- btc_g_results[!is.na(btc_g_results)]
btc_g_results
```

```
> [1] "BTC-USD" "ETH-USD" "BKNG"    "AZO"     "CMG"     "MTD"     "NVR"
```

We will now use these tickers to form the predictor variables of our model.

Fitting a SVR model

Now we have our features we can go ahead and fit a SVM regression model to our data to forecast Bitcoin price. To do this we will need to first import Bitcoin price data aswell as the price data of our predictor variables. However, in order to forecast the Bitcoin price one day ahead we will use price data lagged by one day, to get this data we will use the `import_stonks()` function from our package (SVMForecast).

```
D.1 <- SVMForecast::import_stonks(stock_outcome = c("BTC-USD"),
    stock_pred = btc_g_results, day_lag = c(1))
```

Now we have our dataset we can fit our SVM, but instead of fitting an SVM using a fixed set of hyperparameters we will use tuning. What we do is perform a grid search over hyperparmaters for the best model.

```
T.1 <- SVMForecast::tune_svm(D.1)
```

We now have tuned our parameters and have fitted an SVM to our data, let's see what the performance was and the hyper-parameters that achieved this performance:

```
T.1$best.performance
```

```
> [1] 949136.6
```

```
T.1$best.parameters
```

```
>    gamma cost epsilon
> 54  0.25    8    0.02
```

The performance here was calculated using cross-validation and the MSE as the error function and we note that this number so far doesn't tell us much as we have nothing to compare it to. Later on we will be able to do a quantitative analysis by comparing performance of various models, but for now we will focus on a qualitative analysis to visualize how well our current model may be performing.

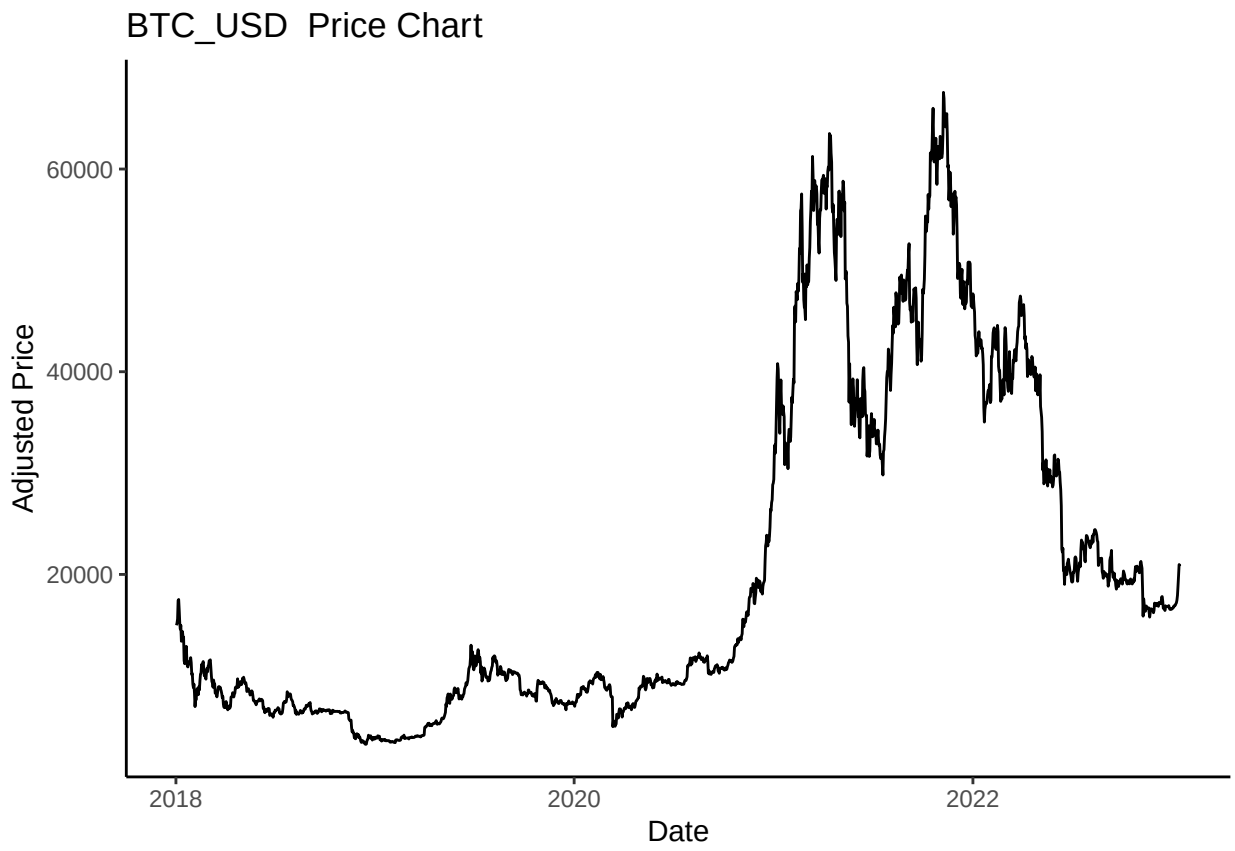
Validating performance

Now we have a model we will check how well its performing by doing some qualitative analysis. Let's first center on a section of time series data and see how our one day ahead forecast compares with what actually happens.

We will use the optimal hyper parameters we found during tuning and fit a model using only a portion of the data (the training data, in black on the figure below), we will then test our model on the remaining data (the testing data, in red in the figure below).

Plot of testing dataset

```
plotp(D.1, "BTC_USD")
```



We now fit our model and find out the number of support vectors of the resulting model:

```
test_size <- 10
n_partition <- 4
tt_ind.1 <- SVMForecast::tt_ranges(D.1, test_size, n_partition)

M.1 <- fit_multi(D.1, tt_ind.1, T.1, k_cross = 0)
```

```
>
> Call:
> svm(formula = formula, data = data, type = "eps-regression", kernel = "radial",
>     gamma = gamma, cost = C, epsilon = eps, cross = k_cross)
>
>
> Parameters:
>    SVM-Type:  eps-regression
>  SVM-Kernel:  radial
>        cost:  8
>       gamma:  0.25
>     epsilon:  0.02
>
>
> Number of Support Vectors:  323
>
>
> Call:
> svm(formula = formula, data = data, type = "eps-regression", kernel = "radial",
```

```
>     gamma = gamma, cost = C, epsilon = eps, cross = k_cross)
>
>
> Parameters:
>    SVM-Type:  eps-regression
>  SVM-Kernel:  radial
>        cost:  8
>       gamma:  0.25
>     epsilon:  0.02
>
>
> Number of Support Vectors:  395
>
>
> Call:
> svm(formula = formula, data = data, type = "eps-regression", kernel = "radial",
>     gamma = gamma, cost = C, epsilon = eps, cross = k_cross)
>
>
> Parameters:
>    SVM-Type:  eps-regression
>  SVM-Kernel:  radial
>        cost:  8
>       gamma:  0.25
>     epsilon:  0.02
>
>
> Number of Support Vectors:  304
>
>
> Call:
> svm(formula = formula, data = data, type = "eps-regression", kernel = "radial",
>     gamma = gamma, cost = C, epsilon = eps, cross = k_cross)
>
>
> Parameters:
>    SVM-Type:  eps-regression
>  SVM-Kernel:  radial
>        cost:  8
>       gamma:  0.25
>     epsilon:  0.02
>
>
> Number of Support Vectors:  308
```

```
for (i in M.1) {
    summary(i)
}
```

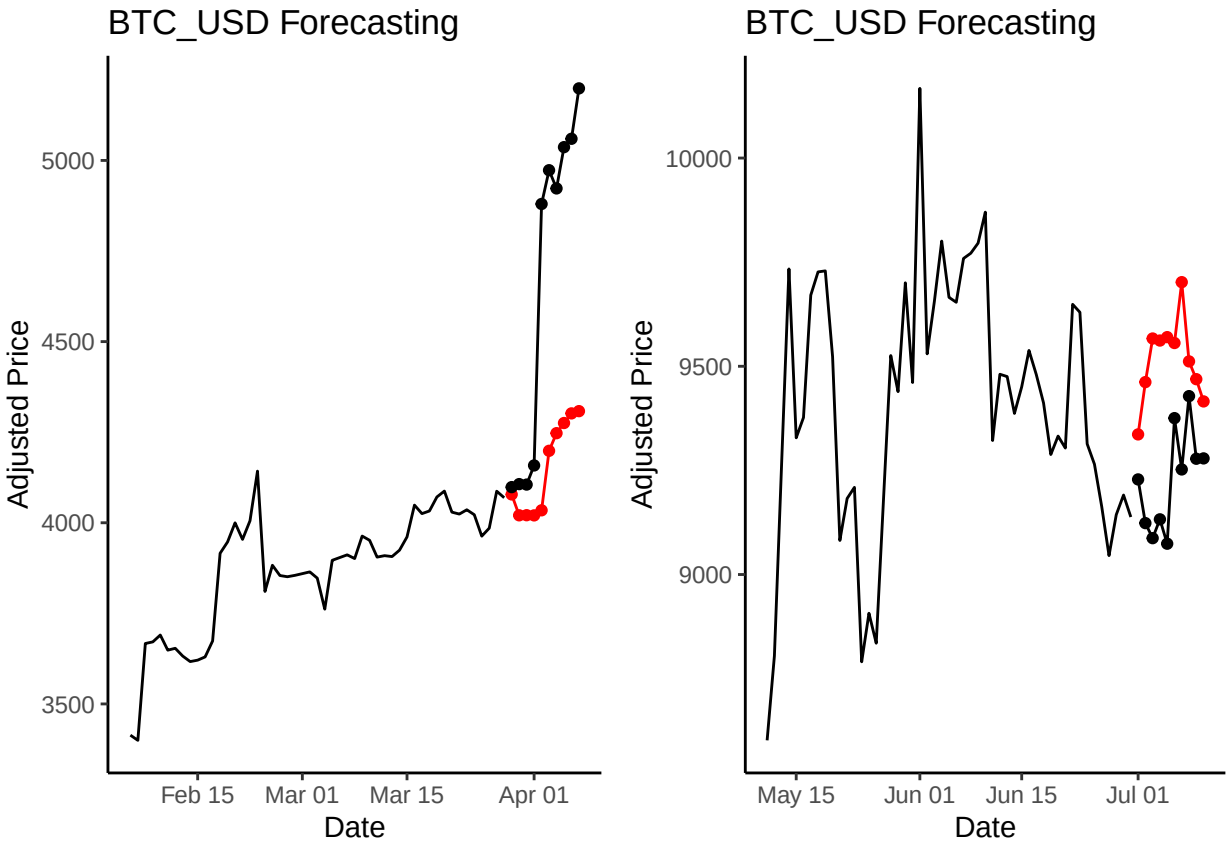
The number of support vectors is large, nearly matching the number of datapoints, which in the case of SVM regression is a sign we have a model which uses a majority of the points to fit the model. It also means the compu. Now let's find our one day ahead forecast predictions:

```
P.1 <- pred_multi(D.1, tt_ind.1, M.1)
```

```
for (i in P.1) {
    print(i)
}
```

Let's now plot the last 50 points of our training data along with our targets and predictors:

```
Plts.1 <- list()
for (i in 1:2) {
    Plts.1[[i]] <- plotf(D.1, tt_ind.1[[i]]$train, tt_ind.1[[i]]$test,
        P.1[[i]], restrict_train = 50)
}
cowplot::plot_grid(plotlist = Plts.1)
```



We see mixed performance here, on the plot on the left the first one day ahead forecast is very close to our target, before our forecasts being to increasingly diverge from the targets. In the plot on the left our predictions get too excited too quickly predicting much higher prices from the get go than the targets.

Playing around with features

Now we have a working model and have visualized its performance we will now play around with the feature space and see if we can create a better model. In particular we want to add or remove features that we think may lead to better modelling. Once we've done this we will tune and fit the models before performing a qualitative analysis to asses their individual performance and a quantitative analysis to compare their performance.

Including past 3 days data:

```
D.3 <- SVMForecast::import_stonks(stock_outcome = c("BTC-USD"),
    stock_pred = btc_g_results, day_lag = c(1, 2, 3))
```

Now we have our dataset we can fit our SVM, but instead of fitting an SVM using a fixed set of hyperparameters we will use tuning. What we do is perform a grid search over hyperparmaters for the best model.

```
T.3 <- SVMForecast::tune_svm(D.3)
```

We now have tuned our parameters and have fitted an SVM to our data, let's see what the performance was and the hyper-parameters that achieved this performance:

```
T.3$best.performance
```

```
> [1] 1059988
```

```
T.3$best.parameters
```

```
>    gamma cost epsilon
> 45 0.125    2    0.02
```

```
plotp(D.1, "BTC_USD")
```

We now fit our model and find out the number of support vectors of the resulting model:

```
test_size <- 10
n_partition <- 4
tt_ind.3 <- SVMForecast::tt_ranges(D.3, test_size, n_partition)

M.3 <- fit_multi(D.3, tt_ind.3, T.3)
```

```
>
> Call:
> svm(formula = formula, data = data, type = "eps-regression", kernel = "radial",
>     gamma = gamma, cost = C, epsilon = eps, cross = k_cross)
>
>
> Parameters:
>    SVM-Type:  eps-regression
>  SVM-Kernel:  radial
>        cost:  2
>       gamma:  0.125
>     epsilon:  0.02
>
>
> Number of Support Vectors:  326
>
>
> Call:
> svm(formula = formula, data = data, type = "eps-regression", kernel = "radial",
```

```
>     gamma = gamma, cost = C, epsilon = eps, cross = k_cross)
>
>
> Parameters:
>    SVM-Type:  eps-regression
>  SVM-Kernel:  radial
>        cost:  2
>       gamma:  0.125
>     epsilon:  0.02
>
>
> Number of Support Vectors:  391
>
>
> Call:
> svm(formula = formula, data = data, type = "eps-regression", kernel = "radial",
>     gamma = gamma, cost = C, epsilon = eps, cross = k_cross)
>
>
> Parameters:
>    SVM-Type:  eps-regression
>  SVM-Kernel:  radial
>        cost:  2
>       gamma:  0.125
>     epsilon:  0.02
>
>
> Number of Support Vectors:  330
>
>
> Call:
> svm(formula = formula, data = data, type = "eps-regression", kernel = "radial",
>     gamma = gamma, cost = C, epsilon = eps, cross = k_cross)
>
>
> Parameters:
>    SVM-Type:  eps-regression
>  SVM-Kernel:  radial
>        cost:  2
>       gamma:  0.125
>     epsilon:  0.02
>
>
> Number of Support Vectors:  337
```

```
for (i in M.3) {
    summary(i)
}
```

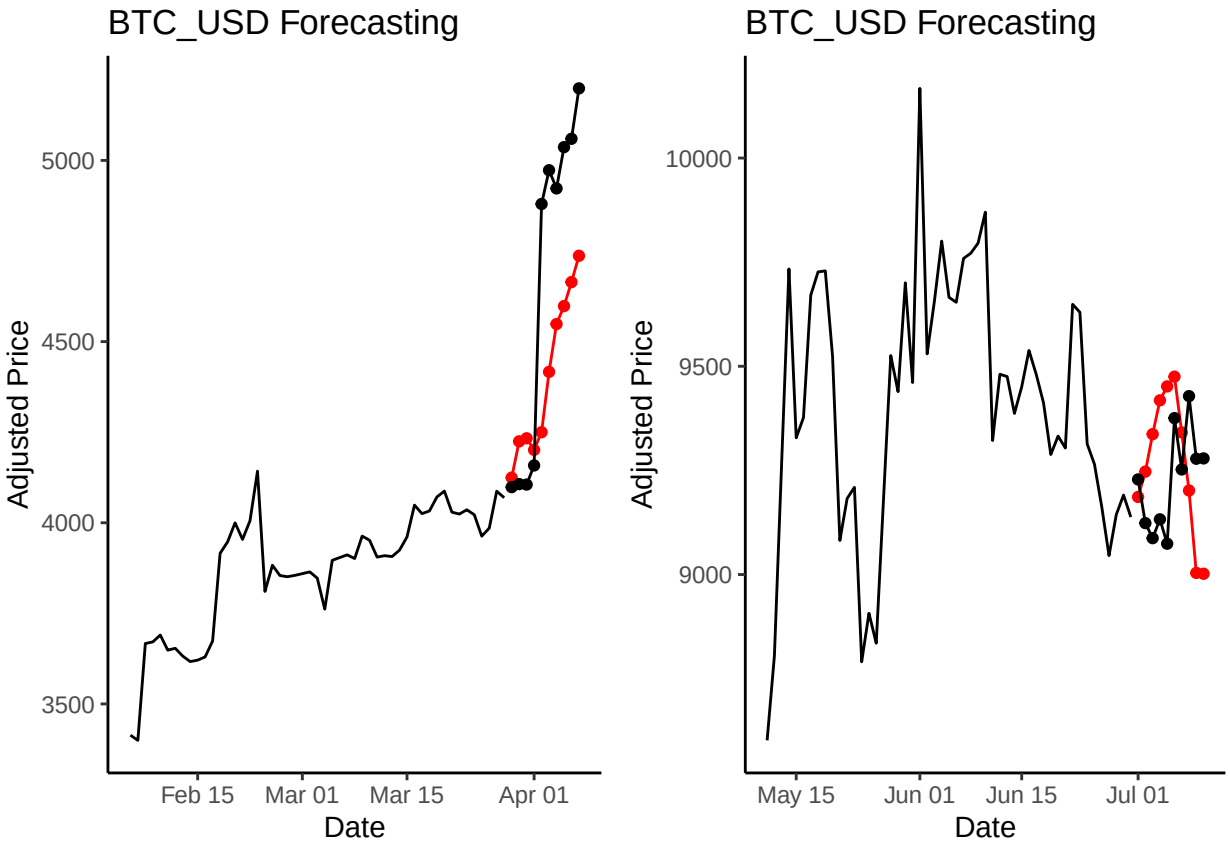
The number of support vectors is large, nearly matching the number of datapoints, which in the case of SVM regression is a sign we have a model which uses a majority of the points to fit the model. It also means the compu. Now let's find our one day ahead forecast predictions:

```
P.3 <- pred_multi(D.3, tt_ind.3, M.3)
```

```
for (i in P.3) {
    print(i)
}
```

Let's now plot the last 50 points of our training data along with our targets and predictors:

```
Plts.3 <- list()
for (i in 1:2) {
    Plts.3[[i]] <- plotf(D.3, tt_ind.3[[i]]$train, tt_ind.3[[i]]$test,
        P.3[[i]], restrict_train = 50)
}
cowplot::plot_grid(plotlist = Plts.3)
```



using crypto currency prices

Will try running a model using a set of cryptocurrencies stocks.

```
stock_preds = c("BTC-USD", "ETH-USD", "BNB-USD", "USDT-USD",
    "ADA-USD", "DOGE-USD", "XRP-USD", "LTC-USD", "TSLA",
    "NVDA", "AMD")
```

Bitcoin price data aswell as the price data of our predictor variables. However, in order to forecast the Bitcoin price one day ahead we will use price data lagged by one day, to get this data we will use the `import_stonks()` function from our package (SVMForecast).

```
D.c <- SVMForecast::import_stonks(stock_outcome = c("BTC-USD"),
    stock_pred = stock_preds, day_lag = c(1, 2, 3))
```

Now we have our dataset we can fit our SVM, but instead of fitting an SVM using a fixed set of hyperparameters we will use tuning. What we do is perform a grid search over hyperparmaters for the best model.

```
T.c <- SVMForecast::tune_svm(D.c)
```

We now have tuned our parameters and have fitted an SVM to our data, let's see what the performance was and the hyper-parameters that achieved this performance:

```
T.c$best.performance
```

```
> [1] 2943489
```

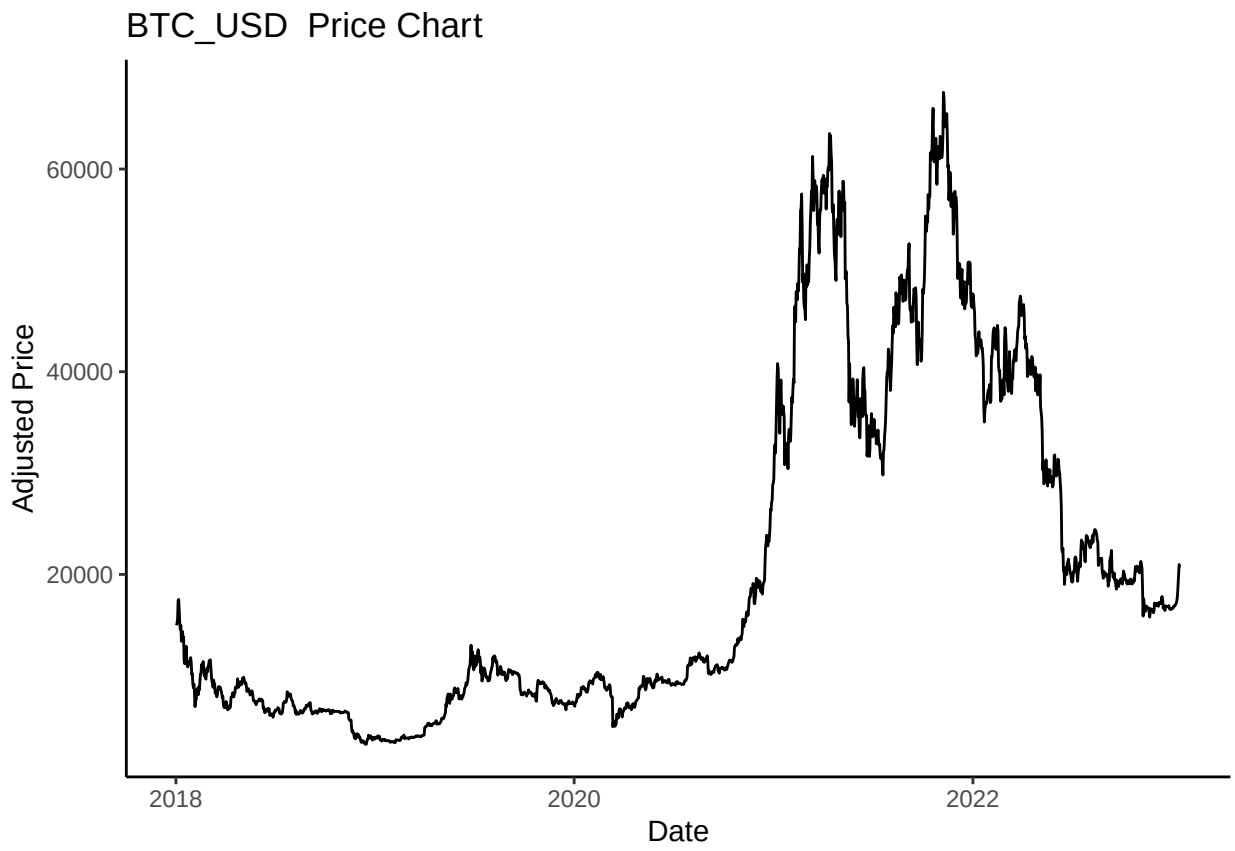
```
T.c$best.parameters
```

```
>   gamma cost epsilon
> 5 0.125    2       0
```

The performance here was calculated using cross-validation and the MSE as again.

Plot of testing dataset

```
plotp(D.c, "BTC_USD")
```



We now fit our model and find out the number of support vectors of the resulting model:

```
test_size <- 10
n_partition <- 4
tt_ind.c <- SVMForecast::tt_ranges(D.c, test_size, n_partition)

M.c <- fit_multi(D.c, tt_ind.c, T.c, k_cross = 0)
```

```
>
> Call:
> svm(formula = formula, data = data, type = "eps-regression", kernel = "radial",
>     gamma = gamma, cost = C, epsilon = eps, cross = k_cross)
>
>
> Parameters:
>    SVM-Type:  eps-regression
>  SVM-Kernel:  radial
>        cost:  2
>       gamma:  0.125
>     epsilon:  0
>
>
> Number of Support Vectors:  448
>
>
> Call:
> svm(formula = formula, data = data, type = "eps-regression", kernel = "radial",
```

```
>     gamma = gamma, cost = C, epsilon = eps, cross = k_cross)
>
>
> Parameters:
>    SVM-Type:  eps-regression
>  SVM-Kernel:  radial
>        cost:  2
>       gamma:  0.125
>     epsilon:  0
>
>
> Number of Support Vectors:  450
>
>
> Call:
> svm(formula = formula, data = data, type = "eps-regression", kernel = "radial",
>     gamma = gamma, cost = C, epsilon = eps, cross = k_cross)
>
>
> Parameters:
>    SVM-Type:  eps-regression
>  SVM-Kernel:  radial
>        cost:  2
>       gamma:  0.125
>     epsilon:  0
>
>
> Number of Support Vectors:  450
>
>
> Call:
> svm(formula = formula, data = data, type = "eps-regression", kernel = "radial",
>     gamma = gamma, cost = C, epsilon = eps, cross = k_cross)
>
>
> Parameters:
>    SVM-Type:  eps-regression
>  SVM-Kernel:  radial
>        cost:  2
>       gamma:  0.125
>     epsilon:  0
>
>
> Number of Support Vectors:  450
```

```
for (i in M.c) {
    summary(i)
}
```

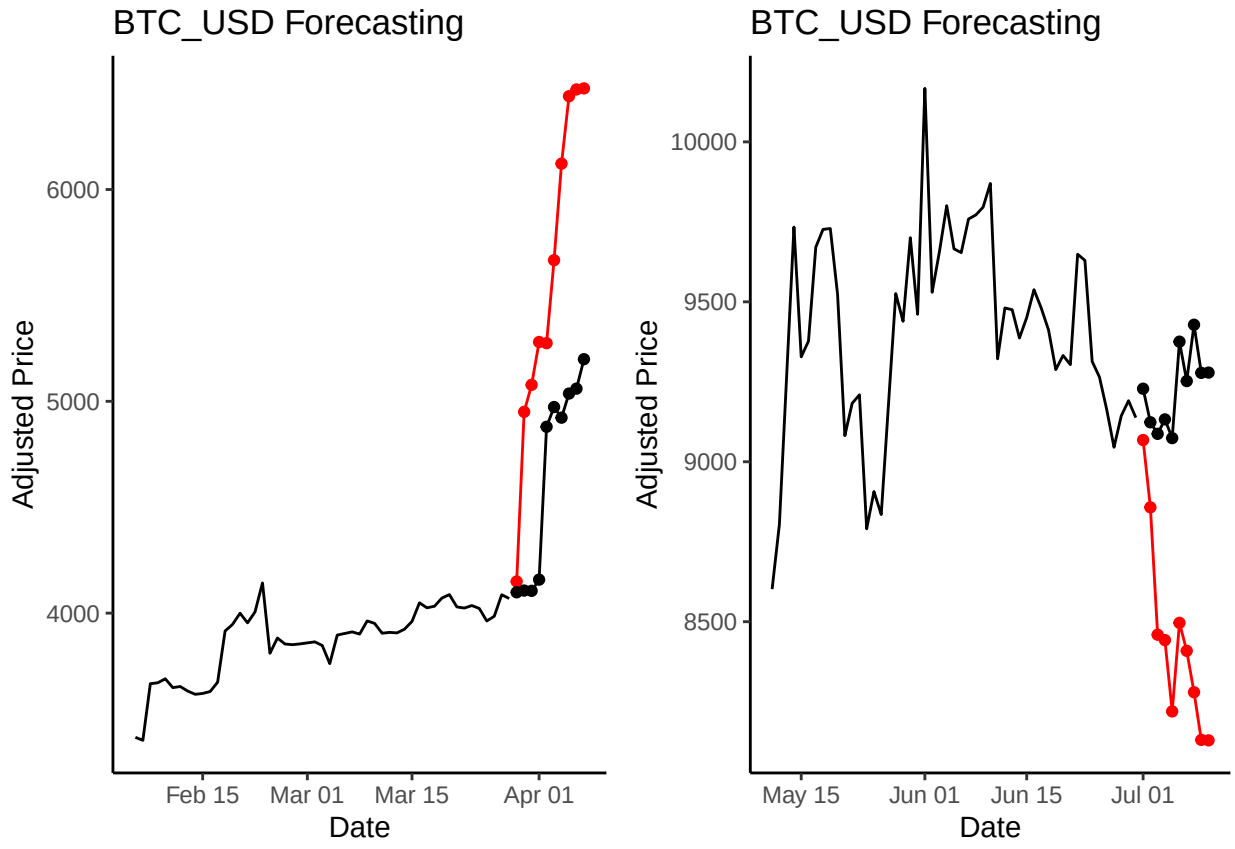
The number of support vectors is large, nearly matching the number of datapoints, which in the case of SVM regression is a sign we have a model which uses a majority of the points to fit the model. It also means the compu. Now let's find our one day ahead forecast predictions:

```
P.c <- pred_multi(D.c, tt_ind.c, M.c)
```

```
for (i in P.c) {
    print(i)
}
```

Let's now plot the last 50 points of our training data along with our targets and predictors:

```
Plts.c <- list()
for (i in 1:2) {
    Plts.c[[i]] <- plotf(D.c, tt_ind.c[[i]]$train, tt_ind.c[[i]]$test,
        P.c[[i]], restrict_train = 50)
}
cowplot::plot_grid(plotlist = Plts.c)
```



A Comparison