SC1 Group Project - Lab Notebook

Henry Bourne, Sam Perren, Dylan Dijk

2023-01-13

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 regression model using our predictor variables.

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

The first section of this report will thus be divided into three sections explaining how we went about performing steps 1, 2 and 3. Later we will.. #TODO: add what else we did and finish off this intro

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 a Least Absolute Shrinkage and Selection Operator (LASSO) method 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.

```r
# We load all the tickers we would like to use for
# analysis
SP500 <- tidyquant::tq_index("SP500")
```

```
## Getting holdings for SP500
```

```
SP500_symbols <- SP500$symbol[1:100]
tickers <- unique(c("BTC-USD", "ETH-USD", "BNB-USD", "USDT-USD",
    "ADA-USD", "DOGE-USD", "XRP-USD", "LTC-USD", "TSLA",
    "NVDA", "AMD", "PYPL", "DJGSP", "FENY", "DJCI", SP500_symbols))

# We fetch and format the data
data <- tidyquant::tq_get(tickers, from = "2019-01-01")
```

```
## Warning: x = 'DJGSP', get = 'stock.prices': Error in getSymbols.yahoo(Symbols = "DJGSP", env = <envi
## DJGSP download failed after two attempts. Error message:
## HTTP error 404.
##  Removing DJGSP.
```

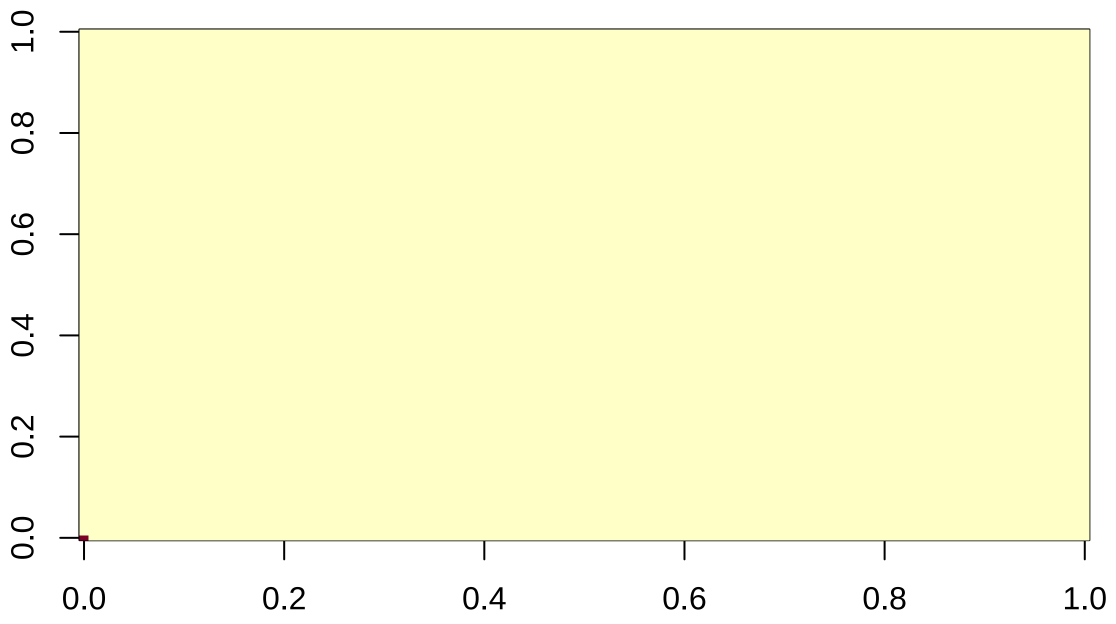
```
data <- data[, c("symbol", "date", "adjusted")]
data <- data %>%
    tidyr::pivot_wider(names_from = symbol, values_from = adjusted)  # Make price coming from each tick
data <- data[, -1]  # get rid of dates column
```

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")
```

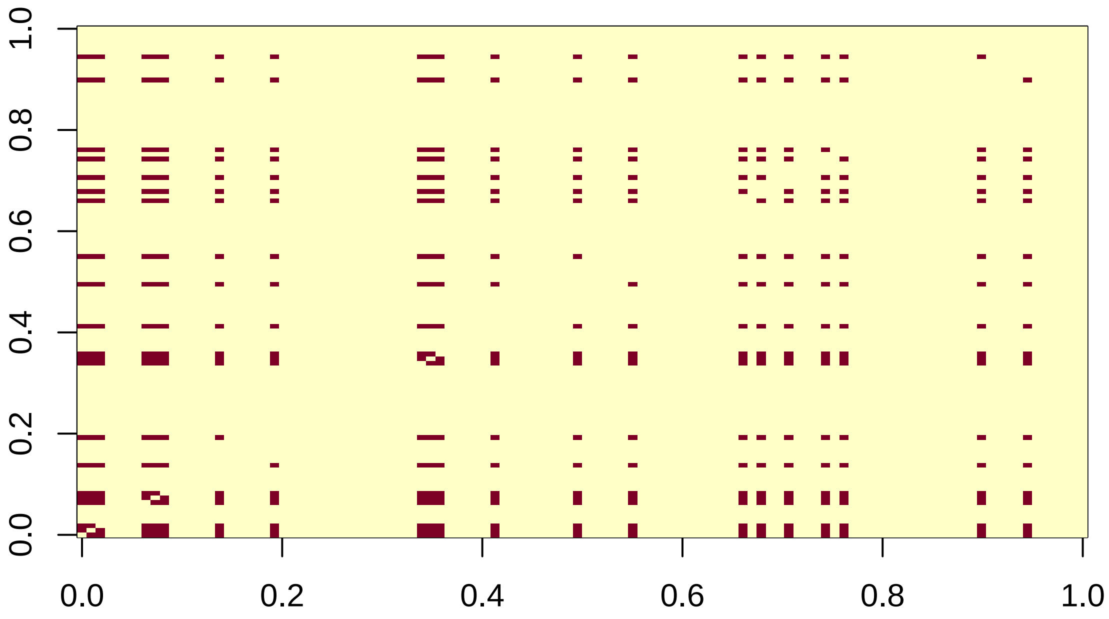
Now we can use *glasso* from the glasso package to use lasso techniques to estimate a sparse covariance matrix, we can plot the resulting covariance matrix:

```
g_results <- glasso::glasso(covs, rho = 9e+05)
# corrplot(g_results$w, is.corr = FALSE,
# method='circle')
image(g_results$w)
```



From the plot we can see that due to high correlations between Bitcoin with itself and Ethereum it is hard to visualize the covariances. What we do now is set the variances to zero as we are not interested in their values and only plot whether a value is zero or one

```r
non_zero <- g_results$w
diag(non_zero) <- 0
non_zero[non_zero > 0] <- 1
non_zero[non_zero < 0] <- 1
# corrplot(non_zero, is.corr = FALSE, method='circle')
image(non_zero)
```



Here we get a more accurate picture of the magnitude of the covariance of the stocks with each other. Let's now look specifically 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" "BNB-USD" "LTC-USD" "TSLA"    "NVDA"    "AAPL"
##  [8] "XOM"     "KO"      "AVGO"    "TMO"     "DIS"     "NKE"     "RTX"
## [15] "SCHW"    "INTC"    "BA"      "ELV"     "PLD"     "TMUS"    "GE"
## [22] "SO"
```

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

Fitting a regression 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] 1031901
```

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

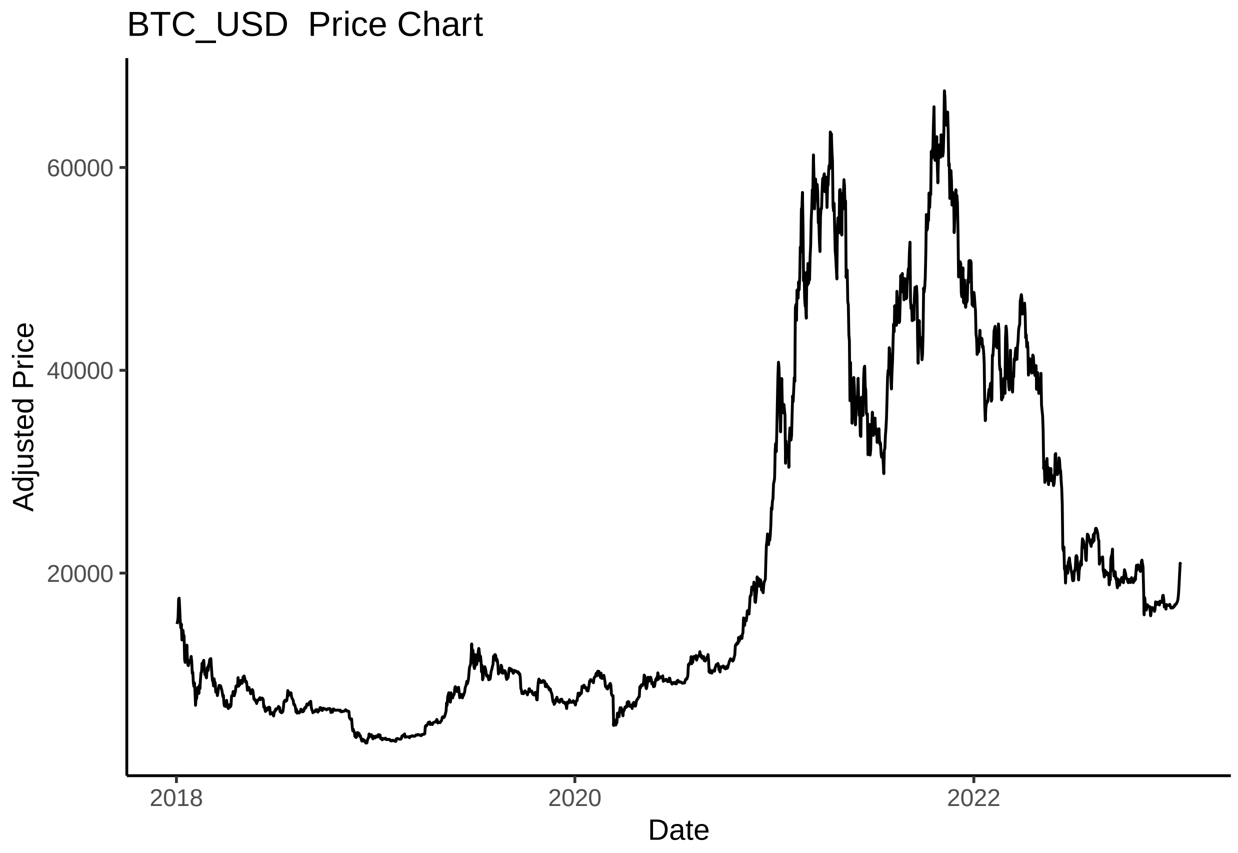
```
##    gamma cost epsilon
## 45 0.125    2    0.02
```

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. #TODO: maybe widen parameters considered in grid search? ie. change defaults

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 hyperparameters 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).

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



We now fit our model and find out the number of support vectors of the resulting model: #TODO: Fiddle with amount of training data, number to leave out and forecast ahead etc.

```r
test_size <- 10
n_partition <- 3
tt_ind.1 <- SVMForecast::tt_ranges(D.1, test_size, n_partition)

# M.1.1 <- fit_svm(D.1.tt[1]$train, gamma=
# T.1$best.parameters$gamma,
# C=T.1$best.parameters$cost,
# eps=T.1$best.parameters$epsilon, k_cross=10)
# summary(M.1)
M.1 <- fit_multi(D.1, tt_ind.1, T.1)
```

```
##
## 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:  462
##
##
## 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:  278
##
##
## 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:  462
```

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

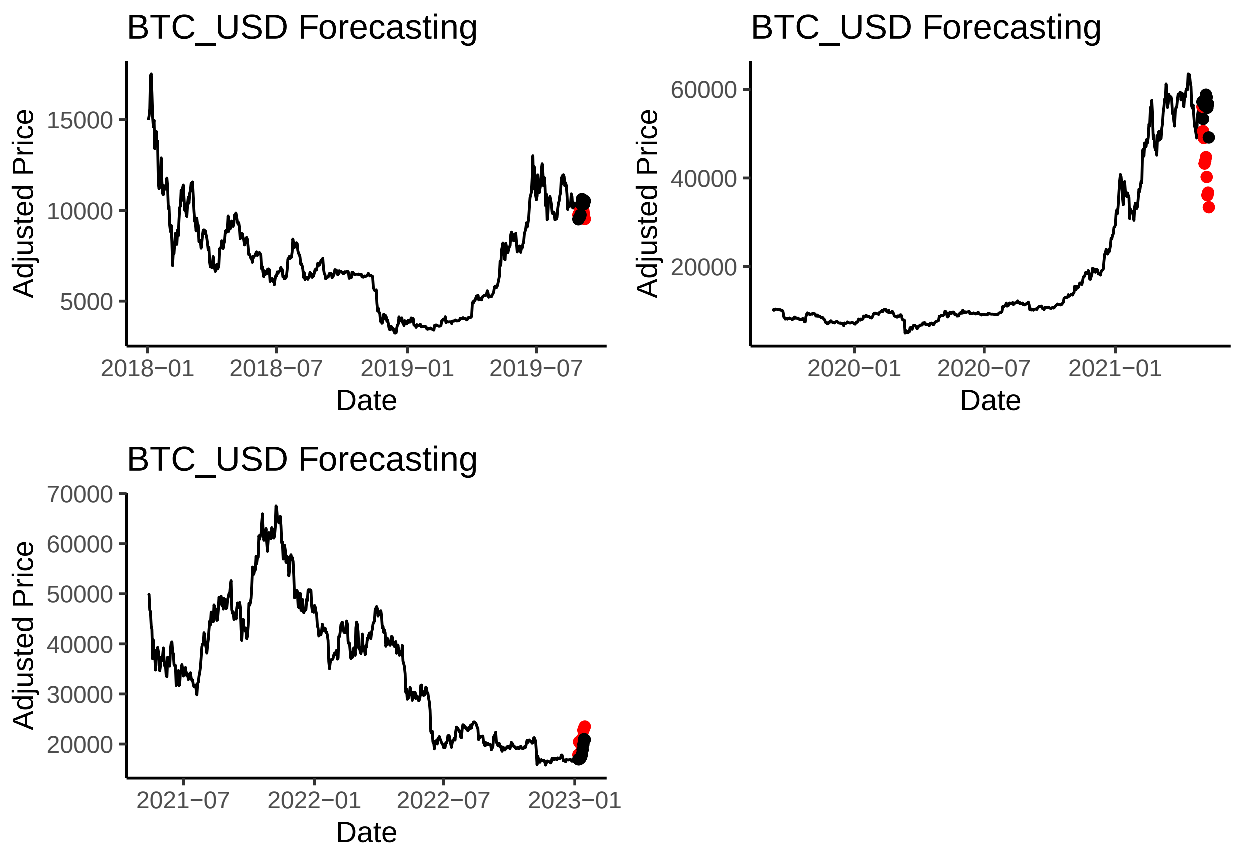
The number of support vectors is large, nearly matching the number of datapoints, which in the case of SVM regression is a sign we have a model which fits the data well. Now let's find our one day ahead forecast predictions:

```r
# pr <- SVMForecast::predict_svm(M.1,
# D.1[test_range,])
P.1 <- pred_multi(D.1, tt_ind.1, M.1)
for (i in P.1) {
    print(i)
}
```

```
##              BTC_USD       Date
## 2019-08-29  9757.356 2019-08-29
## 2019-08-30  9825.926 2019-08-30
## 2019-08-31  9949.557 2019-08-31
## 2019-09-01  9911.616 2019-09-01
## 2019-09-02  9972.769 2019-09-02
## 2019-09-03 10210.771 2019-09-03
## 2019-09-04 10243.317 2019-09-04
## 2019-09-05  9975.893 2019-09-05
## 2019-09-06  9788.896 2019-09-06
## 2019-09-07  9528.246 2019-09-07
##             BTC_USD       Date
## 2021-05-03 56110.52 2021-05-03
## 2021-05-04 50571.88 2021-05-04
## 2021-05-05 49017.18 2021-05-05
## 2021-05-06 43247.94 2021-05-06
## 2021-05-07 43833.17 2021-05-07
## 2021-05-08 44694.50 2021-05-08
## 2021-05-09 40225.67 2021-05-09
## 2021-05-10 36097.85 2021-05-10
## 2021-05-11 36689.70 2021-05-11
## 2021-05-12 33390.64 2021-05-12
##             BTC_USD       Date
## 2023-01-06 17869.47 2023-01-06
## 2023-01-07 20442.85 2023-01-07
## 2023-01-08 20446.88 2023-01-08
## 2023-01-09 20509.14 2023-01-09
## 2023-01-10 19796.16 2023-01-10
## 2023-01-11 20384.92 2023-01-11
## 2023-01-12 21182.67 2023-01-12
## 2023-01-13 22740.77 2023-01-13
## 2023-01-14 23171.93 2023-01-14
## 2023-01-15 23491.28 2023-01-15
```

```r
# plt <- plotf(D.1, tt_ind.1[[1]]$train,
# tt_ind.1[[1]]$test, P.1[[1]])
```

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



#TODO: generate test and training range #TODO: predict and return in desired format #TODO: plot training data and predicted and target points

Playing around with features

Selecting features

#TODO: formulate subsets of tickers that you are going to use to fit svm's, with reasoning

Fitting models

#TODO: fit svms using tuning

Qualative analysis

#TODO: plot qualative analysis

A Comparison

#TODO: perform quantative analysis comparing performance between them