SC1 Group Project - Lab Notebook

Henry Bourne, Sam Perren, Dylan Dijk

2023-01-13

```
devtools::load_all(path = "SVMForecast/")
```

```
> i Loading SVMForecast
```

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).

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 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
```

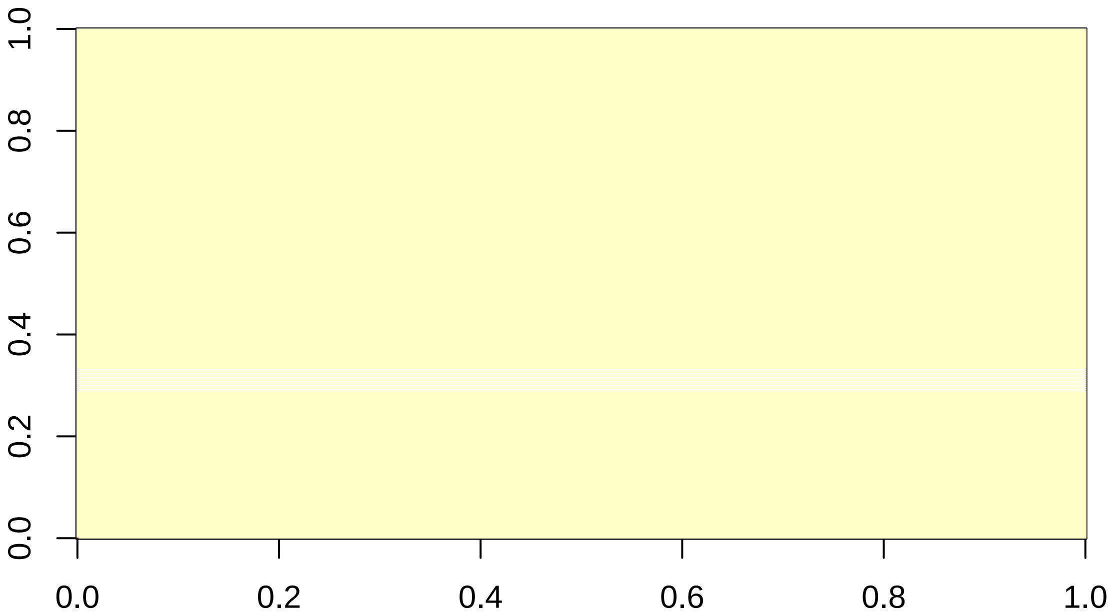
```r
SP500_symbols <- SP500$symbol
tickers <- unique(c("BTC-USD", "ETH-USD", "BNB-USD", "USDT-USD",
    "ADA-USD", "DOGE-USD", "XRP-USD", "LTC-USD", "TSLA",
    "NVDA", "AMD", "PYPL", SP500_symbols))



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

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

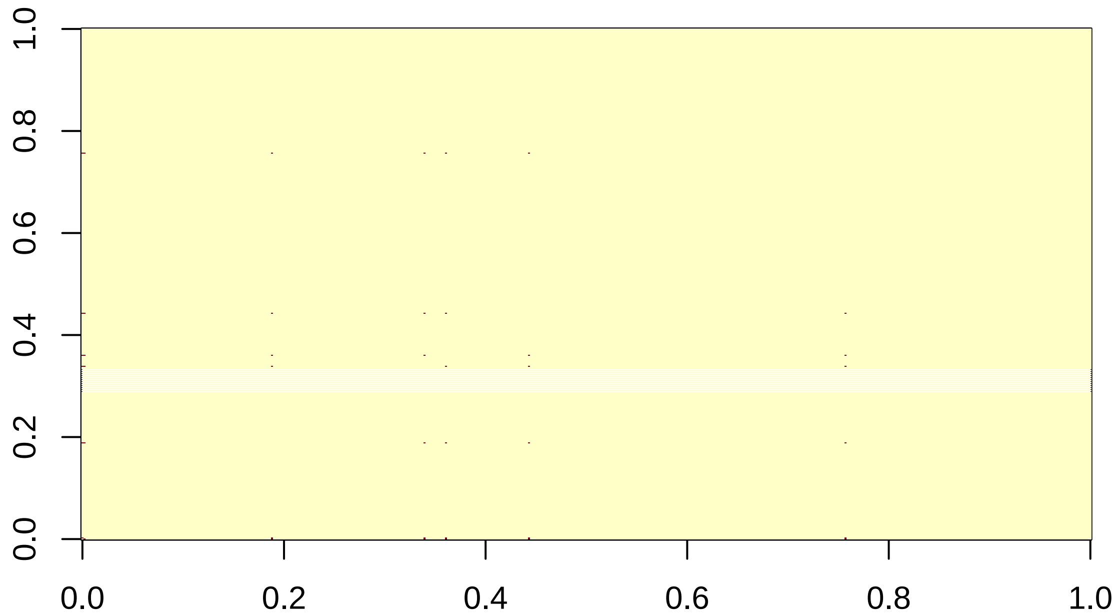
Now we can use *glasso* from the glasso package to use lasso techniques to estimate a sparse covariance matrix, we can plot the resulting covariance matrix:

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

```
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" "BKNG"    "AZO"     "CMG"     "MTD"     "NVR"
```

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:

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

```
> [1] 949136.6
```

```r
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 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).

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

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

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

```r
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:  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
```

```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 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:

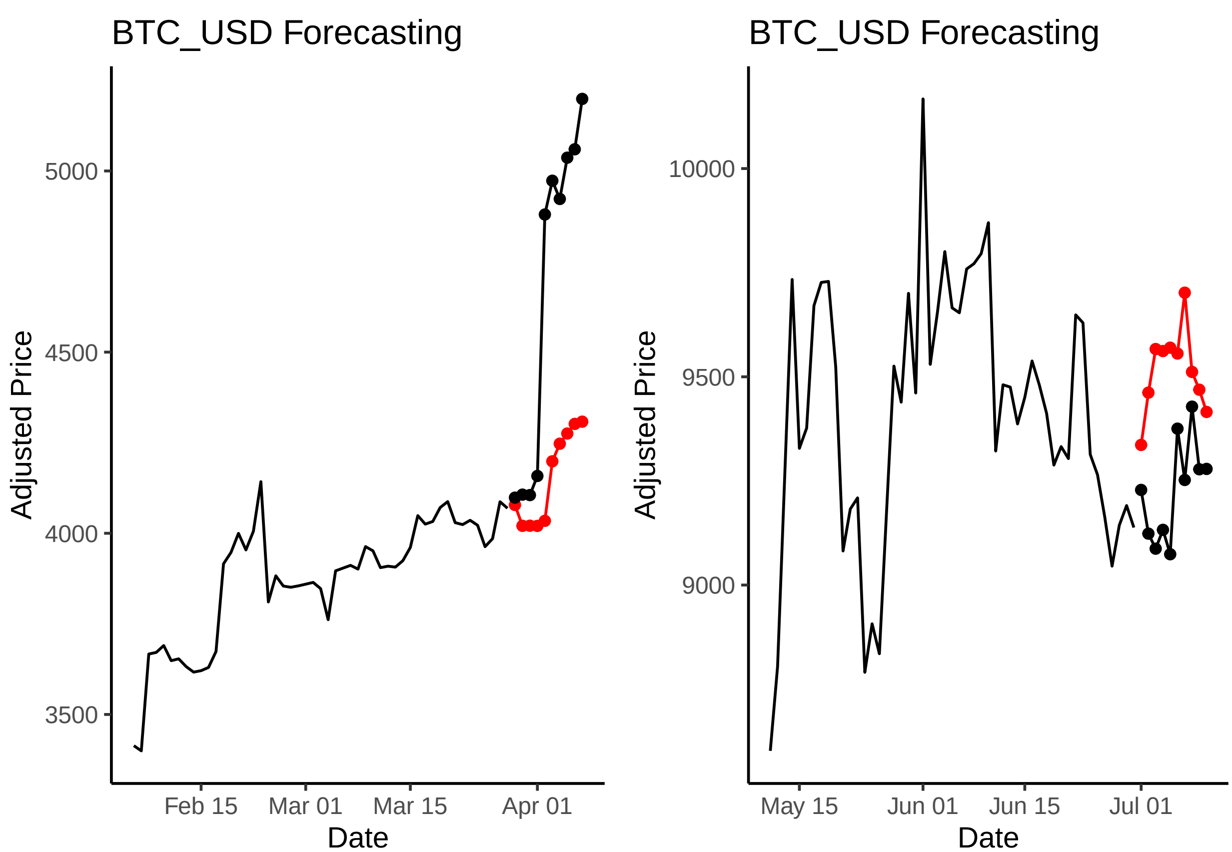
```r
P.1 <- pred_multi(D.1, tt_ind.1, M.1)
for (i in P.1) {
    print(i)
}
```

```
>             BTC_USD       Date
> 2019-03-29 4078.190 2019-03-29
> 2019-03-30 4020.589 2019-03-30
> 2019-03-31 4020.897 2019-03-31
> 2019-04-01 4020.371 2019-04-01
> 2019-04-02 4034.407 2019-04-02
> 2019-04-03 4198.605 2019-04-03
> 2019-04-04 4247.243 2019-04-04
> 2019-04-05 4275.292 2019-04-05
> 2019-04-06 4301.879 2019-04-06
> 2019-04-07 4308.039 2019-04-07
>             BTC_USD       Date
> 2020-07-01 9336.444 2020-07-01
> 2020-07-02 9461.963 2020-07-02
> 2020-07-03 9566.644 2020-07-03
> 2020-07-04 9561.802 2020-07-04
> 2020-07-05 9569.922 2020-07-05
> 2020-07-06 9555.854 2020-07-06
> 2020-07-07 9701.957 2020-07-07
> 2020-07-08 9511.717 2020-07-08
> 2020-07-09 9469.023 2020-07-09
> 2020-07-10 9415.600 2020-07-10
>             BTC_USD       Date
> 2021-10-04 48271.39 2021-10-04
> 2021-10-05 48850.77 2021-10-05
> 2021-10-06 51064.53 2021-10-06
> 2021-10-07 51572.00 2021-10-07
> 2021-10-08 48985.10 2021-10-08
> 2021-10-09 49671.30 2021-10-09
> 2021-10-10 49958.64 2021-10-10
```

```
> 2021-10-11 49985.40 2021-10-11
> 2021-10-12 51578.70 2021-10-12
> 2021-10-13 51181.64 2021-10-13
>              BTC_USD       Date
> 2023-01-07 18445.90 2023-01-07
> 2023-01-08 18439.43 2023-01-08
> 2023-01-09 18489.89 2023-01-09
> 2023-01-10 18247.70 2023-01-10
> 2023-01-11 18504.40 2023-01-11
> 2023-01-12 19472.25 2023-01-12
> 2023-01-13 19808.04 2023-01-13
> 2023-01-14 21005.80 2023-01-14
> 2023-01-15 21244.46 2023-01-15
> 2023-01-16 21229.46 2023-01-16
```

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

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] 1071010
```

```
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)
```

```r
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
```

```r
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:

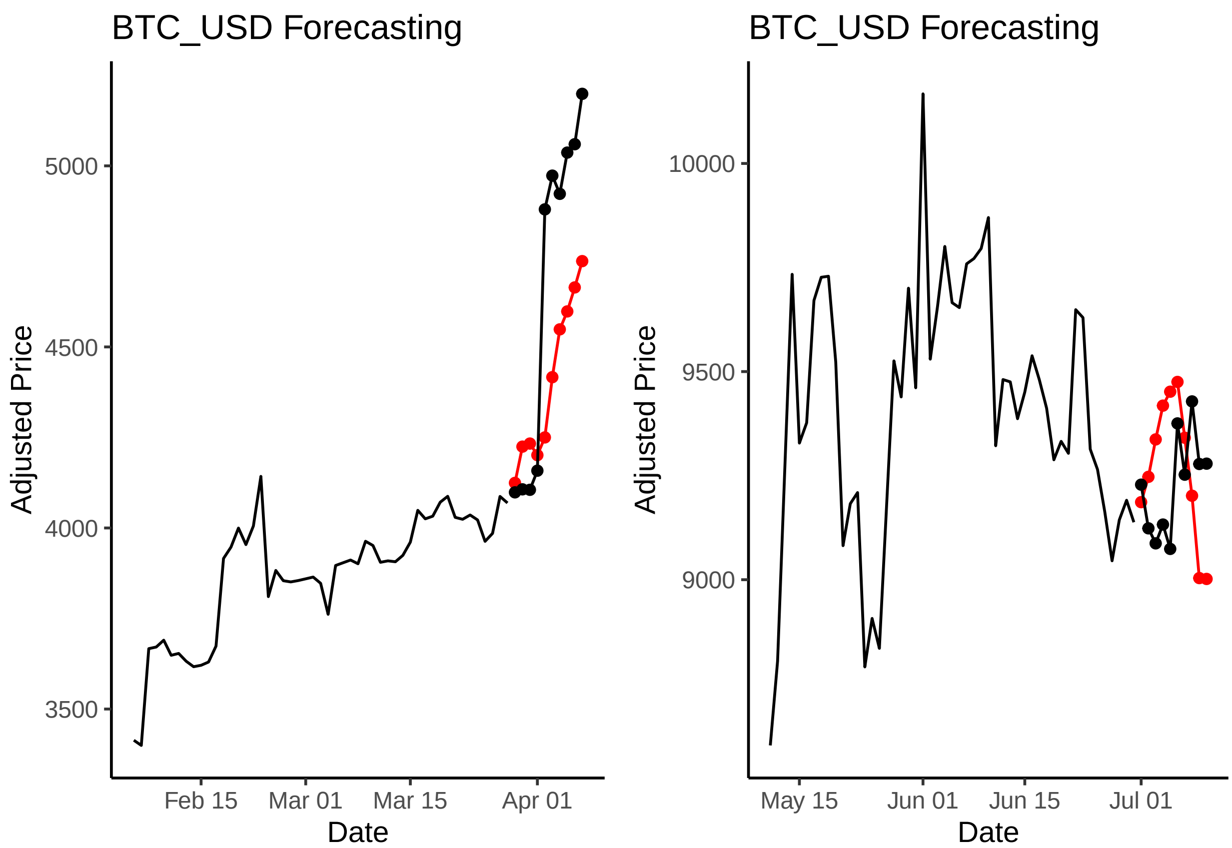
```r
P.3 <- pred_multi(D.3, tt_ind.3, M.3)
for (i in P.3) {
    print(i)
}
```

```
>               BTC_USD       Date
> 2019-03-29 4124.492 2019-03-29
> 2019-03-30 4224.639 2019-03-30
> 2019-03-31 4233.159 2019-03-31
> 2019-04-01 4201.137 2019-04-01
> 2019-04-02 4250.079 2019-04-02
> 2019-04-03 4416.450 2019-04-03
> 2019-04-04 4548.477 2019-04-04
> 2019-04-05 4598.056 2019-04-05
> 2019-04-06 4664.283 2019-04-06
> 2019-04-07 4736.906 2019-04-07
>               BTC_USD       Date
> 2020-07-01 9186.291 2020-07-01
> 2020-07-02 9247.074 2020-07-02
> 2020-07-03 9337.006 2020-07-03
> 2020-07-04 9418.181 2020-07-04
> 2020-07-05 9451.688 2020-07-05
> 2020-07-06 9475.133 2020-07-06
> 2020-07-07 9341.068 2020-07-07
> 2020-07-08 9201.669 2020-07-08
> 2020-07-09 9003.821 2020-07-09
> 2020-07-10 9001.771 2020-07-10
>               BTC_USD       Date
> 2021-10-04 47918.86 2021-10-04
> 2021-10-05 48027.20 2021-10-05
> 2021-10-06 48569.72 2021-10-06
> 2021-10-07 48970.97 2021-10-07
> 2021-10-08 47569.90 2021-10-08
> 2021-10-09 47304.29 2021-10-09
> 2021-10-10 47118.28 2021-10-10
```

```
> 2021-10-11 47295.50 2021-10-11
> 2021-10-12 47693.26 2021-10-12
> 2021-10-13 47443.04 2021-10-13
>              BTC_USD       Date
> 2023-01-07 18698.22 2023-01-07
> 2023-01-08 19056.76 2023-01-08
> 2023-01-09 19628.10 2023-01-09
> 2023-01-10 19701.56 2023-01-10
> 2023-01-11 20009.30 2023-01-11
> 2023-01-12 21068.90 2023-01-12
> 2023-01-13 22172.92 2023-01-13
> 2023-01-14 23776.37 2023-01-14
> 2023-01-15 24959.98 2023-01-15
> 2023-01-16 25628.75 2023-01-16
```

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)
```



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