Strategy development for EUD/USD x GBP/USD

Mission statement

The objective of this task is to build models which yield a profitable set of trades for the GBP/USD FX pair, informed by the movements of the EUR/USD pair. The mantra 'simple models work best' will be adhered to. The strategy will focus on the detection of larger moves in EUR/USD, and the proportional relationship between the equivalent move in GBP/USD at a simultaneous timestep. I am going to try to build a profitable model (or at least a functional one), in the most informed way that I can given the time available (weekend 26-27th Jan).

Model training objective

Statistical learning models will be used to predict price maxima and minima within a time window *t* of a trigger *x.* The ideal model will output a simple estimate of price, with sufficient predictive power to inform a trading strategy with an 'edge'.

Use of Tech/Data

- Language: Python 3.5
- Scikit-learn, for GridSearchCV and modelling functions
- TA-lib python wrapper, for the technical indicator functions
- Bayesian-optimisation library, for optimising black-box functions
- Pandas and Numpy for data manipulation
- Matplotlib for graph generation
- Free FX minute data from 2001-2018 retrieved from https://forextester.com/data/datasources

Model Use

Random Forest Regression models from scikit-learn were used to estimate prices. GridSearchCV was used to tune hyperparameters, with 4-fold time-slice CV. Most often the grid search ended up choosing small numbers of features in decisions per split, and shallow tree depths. The modal minimum-MSE tree found during trigger optimisation (which was ultimately abandoned) had a depth of 4, with 3 random features considered per node split, and ~50 estimators in the ensemble. The final optimised estimator parameter sets are quite like this also (see below)

Another model type which could be tried is Bayesian-ridge regression. The useful thing about BR is that it treats all features as gamma PDFs, given learned probabilities, and the regression fit uses these to generate a standard deviation for each of its estimates. This may allow a more adaptive tuning of stops, given information about the variance of price estimates. This does come with more feature pre-processing requirements than RF, however.

Models NOT used

ARIMA or VAR style timeseries models have not been used. They seem to be the obvious choice for modelling timeseries correlations/regressions, which suggests that everyone is using/trying them, ergo let's not do so. Contrarian attitudes seems to be one personality trait most successful traders have in common (at least as far as I've encountered them in podcast form).

NNets, SVMs, GBMs. These model types are prone to overfit. We could isolate certain market triggers and train very large input hyper-optimised instances of these things. The added complexity of avoiding the overfit and tuning these relatively compute-intensive models doesn't seem worth it for the first attempt.

Predictive Features

ATR-Relative trigger scale: *abs(openEUR – closeEUR) / ATR(20)*

Immediate GBP dollar-proportional move: *(dEUR / openEUR) / (dGBP / openGBP)* at t_0

Log(closeGBP/MASlow(GBP)) at t_0 *(log to preserve distribution symmetry)*

Log(closeGBP/MAFast(GBP)) at t_0

Log(closeEUR/MASlow(EUR)) at t_0

Log(closeEUR/MAFast(EUR)) at t_0

RSI(EURUSD) at t_0

RSI(GBPUSD) at t_0

ATR(EURUSD) at t_0

ATR(GBPUSD) at t_0

##More to be added or removed

The free data availability for volume was not of enough quantity or quality compared to the price data, so volume predictors have not been included.

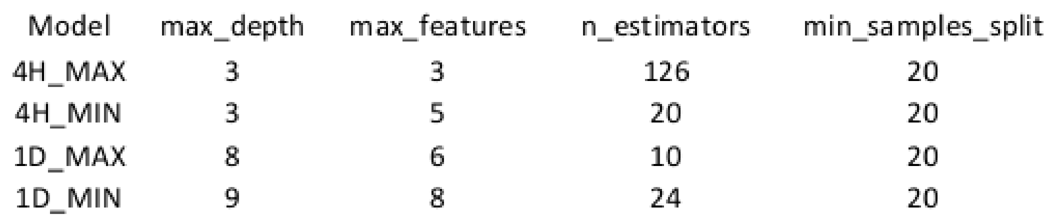
Predicted Features

Where *x* is the trigger index, and *t* is the timestep count.

Maximum price in time window. Max(highGBP [x: t+x])

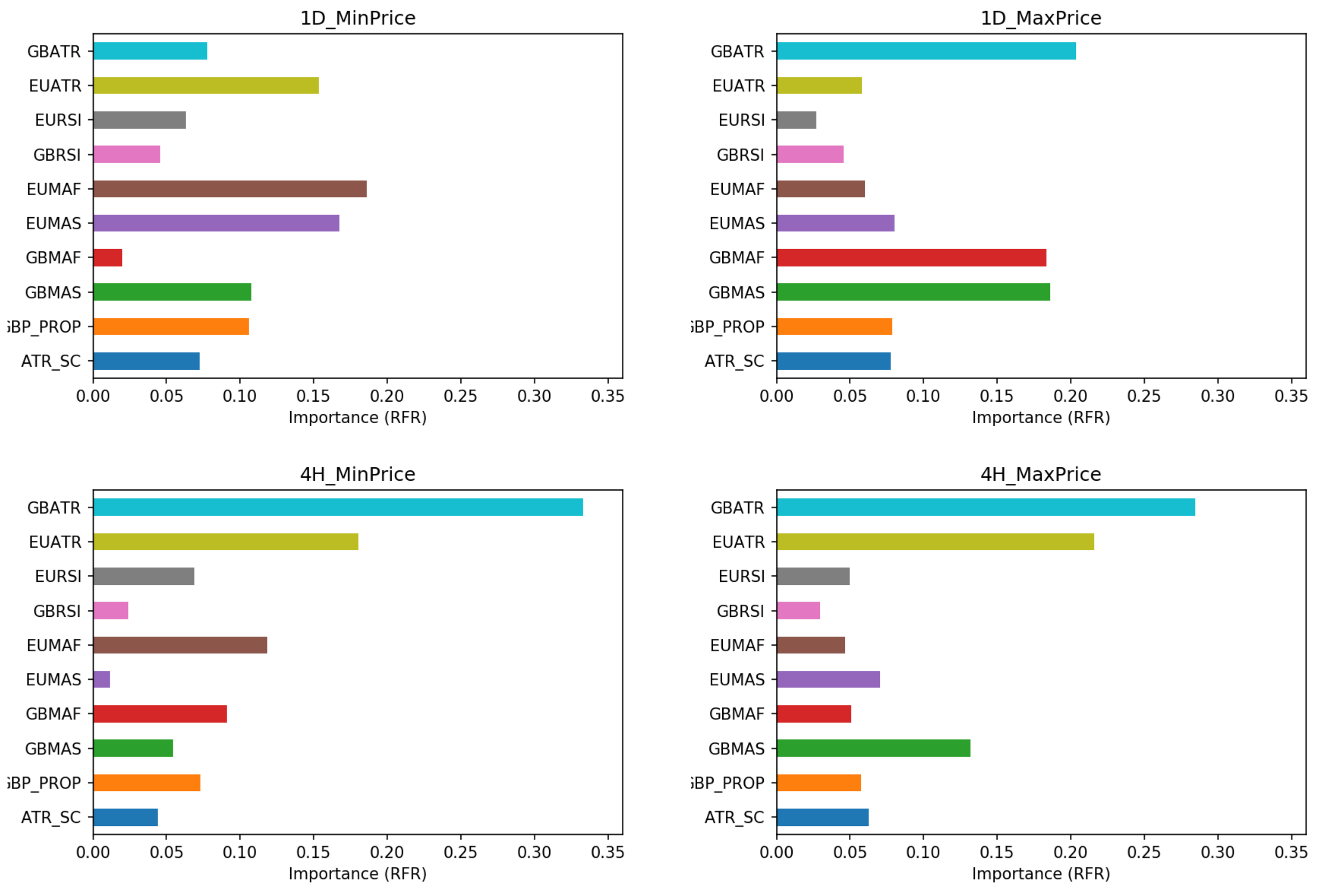
Minimum price in time window. Min(lowGBP [x: t+x])

Modelling outcomes for optimised RFRs



Model	max_depth	max_features	n_estimators	min_samples_split
4H_MAX	3	3	126	20
4H_MIN	3	5	20	20
1D_MAX	8	6	10	20
1D_MIN	9	8	24	20

Optimised hyperparameters for each RFR

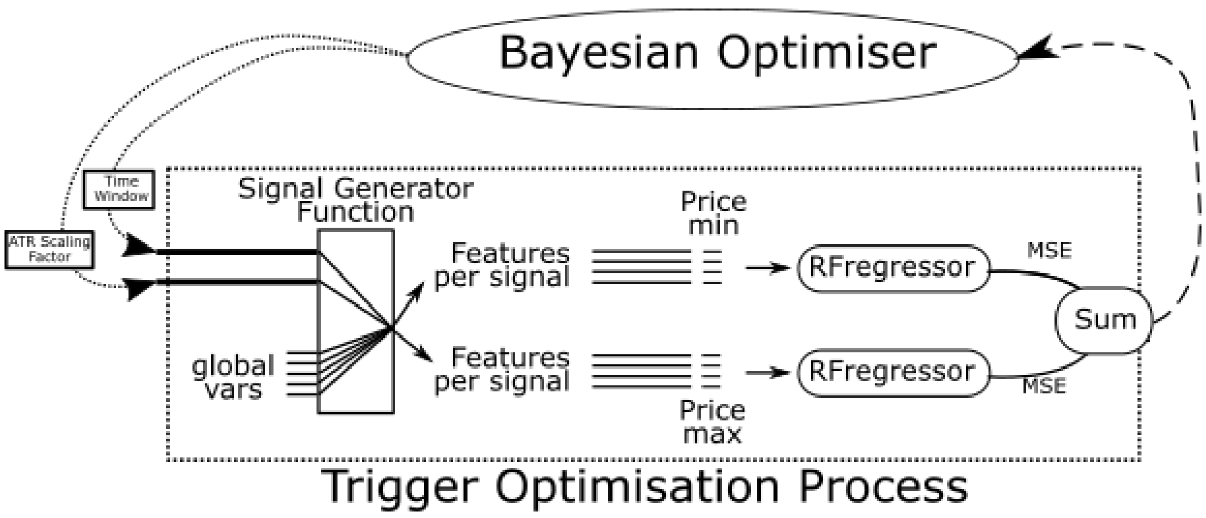


Feature Importances extracted from CV best estimators

It is remarkable that the features differ in importance so much by just changing the timeframe. Although the ATR of GBP does seem to be a regular feature.

Optimisation Loops

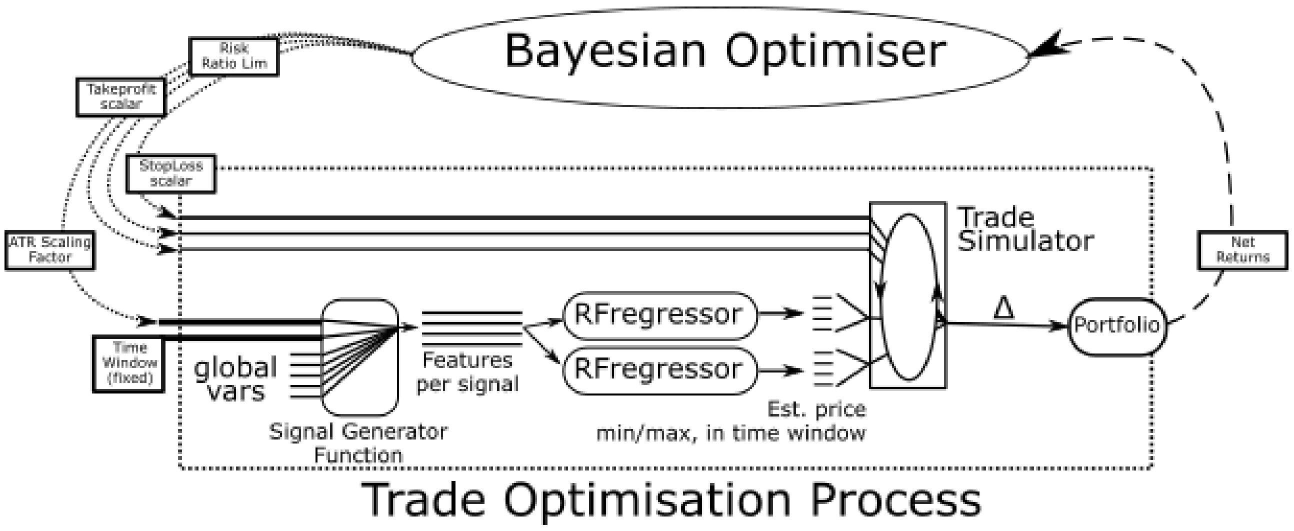
Two applications of the Bayesian ‘black-box’ function optimisation where attempted. The first was an optimisation of the trade entry/signal generation function. The parameters were the ATR scaling factor and the time window before stop. The opt function sought out the pair of price prediction models with the minimum combined MSE. The objective of optimising this function was to find a signal set which gave more predictive power to the model building process, and ergo which could be more reliably trained.



This is an outline of the trigger optimisation

The other application of the Bayesian optimiser was to tune the parameters of the trading system that used the predictive models. The optimiser sought to maximise returns (naturally) and tuned buy and sell stop adjustments, the risk-ratio, and ATR scaling factor again. In the end the risk-ratio was set at 1.2 and excluded from tuning though.

Both optimisation loops were performed on the same training data. The test data (60% of the entire time series) was left aside for testing at the end, this is shown in some graphs below.



This is an outline of the optimisation of the trade system

1D: Bayesian Optimisation of Trigger Function -> Model Predictive Power

The ATRsc parameter controls the scaling of the Average True Range indicator. The trigger occurs if the absolute 1-timestep movement exceeds the product of the ATR and its scaling factor.

The 'twind' parameter controls the time window within which trades are sought following the trigger. In this case a twind of 1 would mean only 1 day.

```
bounds = {'ATRsc' : (0.5,1.3), 'twind' : (1,6)}
```

```
|   iter     |  target   |   ATRsc    |   twind   |
-------------------------------------------------
|  1         | -0.000309 |  0.8336    |  4.602    |
|  2         | -0.000210 |  0.5001    |  2.512    |
|  3         | -0.000125 |  0.6174    |  1.462    |
|  4         | -0.000316 |  1.282     |  5.994    |
|  5         | -0.000122 |  0.8863    |  1.09     |
|  6         | -0.000337 |  1.271     |  5.975    |
|  7         | -0.000121 |  1.263     |  1.053    |
|  8         | -0.000369 |  0.7107    |  5.99     |
|  9         | -0.000122 |  0.7663    |  1.048    |
|  10        | -0.000382 |  0.6116    |  5.985    |
|  11        | -0.000120 |  1.264     |  1.028    |
|  12        | -0.000120 |  1.24      |  1.019    |
|  13        | -0.000396 |  0.9881    |  5.923    |
|  14        | -0.000112 |  1.136     |  1.067    |
|  15        | -0.000126 |  0.8543    |  1.084    |
|  16        | -0.000280 |  1.154     |  5.895    |
|  17        | -0.000124 |  0.9055    |  1.18     |
|  18        | -0.000114 |  0.8008    |  1.013    |
|  19        | -0.000381 |  0.6346    |  5.953    |
|  20        | -0.000125 |  0.6093    |  1.198    |
|  21        | -0.000127 |  1.04      |  1.14     |
|  22        | -0.000328 |  0.7964    |  5.977    |
|  23        | -0.00013  |  0.5409    |  1.035    |
|  24        | -0.000127 |  1.095     |  1.336    |
|  25        | -0.000393 |  0.9694    |  5.992    |
|  26        | -0.000123 |  1.03      |  1.034    |
|  27        | -0.000371 |  0.6743    |  5.997    |
|  28        | -0.000117 |  0.8001    |  1.312    |
|  29        | -0.000126 |  0.6118    |  1.148    |
|  30        | -0.000393 |  0.6089    |  5.807    |
|  31        | -0.000125 |  0.6264    |  1.638    |
|  32        | -0.000334 |  1.067     |  5.957    |
|  33        | -0.000124 |  0.6606    |  1.39     |
=================================================================
```

It just minimises ‘twind’ all the time!

1D: Bayesian Optimisation of Trade Function -> Profit

```
back_bounds = {'Qprofit' : (0.4,1.0), 'Qloss' : (0.9,1.4),
                                'ATRsc' : (0.7,0.9)}

|   iter    |  target   |   ATRsc   |   Qloss   |  Qprofit  |
-------------------------------------------------------------
|  1        |  1.552    |  0.7153   |  1.29     |  0.663    |
|  2        |  1.695    |  0.8447   |  1.389    |  0.7231   |
|  3        |  1.309    |  0.8002   |  0.936    |  0.5611   |
|  4        |  6.822    |  0.8      |  1.24     |  0.8822   |
|  5        |  9.402    |  0.8002   |  1.227    |  0.9015   |
|  6        |  9.388    |  0.7935   |  1.222    |  0.9001   |
|  7        |  10.19    |  0.8025   |  1.217    |  0.9047   |
|  8        |  10.33    |  0.8001   |  1.217    |  0.9159   |
|  9        |  10.34    |  0.8139   |  1.211    |  0.9165   |
|  10       |  10.88    |  0.8024   |  1.197    |  0.9147   |
|  11       |  10.21    |  0.8115   |  1.187    |  0.9052   |
|  12       |  16.23    |  0.8011   |  1.187    |  0.9308   |
|  13       |  16.92    |  0.8036   |  1.186    |  0.9336   |
|  14       |  18.63    |  0.8015   |  1.185    |  0.9371   |
|  15       |  18.65    |  0.7989   |  1.186    |  0.9381   |
|  16       |  19.28    |  0.7992   |  1.183    |  0.9375   |
|  17       |  19.33    |  0.7997   |  1.182    |  0.9399   |
|  18       |  19.29    |  0.7994   |  1.18     |  0.9382   |
|  19       |  19.31    |  0.7974   |  1.181    |  0.9388   |
|  20       |  19.3     |  0.799    |  1.181    |  0.9386   |
|  21       |  19.65    |  0.7989   |  1.178    |  0.9425   |
|  22       |  19.68    |  0.8019   |  1.178    |  0.9438   |
|  23       |  19.74    |  0.7998   |  1.179    |  0.9461   |
|  24       |  21.41    |  0.7997   |  1.176    |  0.9466   |
=============================================================
```

4H: Bayesian Optimisation of Trigger Function -> Model Predictive Power

```
|   iter    |  target   |   ATRsc   |   twind   |
-------------------------------------------------
|  1        | -1.76e-05 |  1.126    |  2.441    |
|  2        | -1.477e-0 |  0.5002   |  1.605    |
|  3        | -9.26e-06 |  1.997    |  1.001    |
|  4        | -1.959e-0 |  0.5747   |  2.989    |
|  5        | -1.037e-0 |  1.973    |  1.024    |
|  6        | -1.433e-0 |  1.74     |  2.946    |
|  7        | -1.455e-0 |  0.6215   |  1.051    |
|  8        | -1.289e-0 |  1.968    |  2.926    |
|  9        | -1.915e-0 |  0.6867   |  2.911    |
|  10       | -1.009e-0 |  1.964    |  1.146    |
|  11       | -1.896e-0 |  0.739    |  2.946    |
|  12       | -9.898e-0 |  1.966    |  1.142    |
|  13       | -1.428e-0 |  0.7064   |  1.128    |
|  14       | -1.898e-0 |  0.746    |  2.789    |
|  15       | -1.983e-0 |  0.5388   |  2.638    |
|  16       | -1.9e-05  |  0.6569   |  2.897    |
|  17       | -1.048e-0 |  1.662    |  1.067    |
|  18       | -1.822e-0 |  0.9937   |  2.796    |
|  19       | -1.225e-0 |  1.207    |  1.751    |
|  20       | -1.163e-0 |  1.4      |  1.116    |
|  21       | -1.52e-05 |  1.826    |  2.99     |
|  22       | -1.151e-0 |  1.399    |  1.136    |
|  23       | -1.636e-0 |  1.598    |  2.147    |
|  24       | -1.481e-0 |  0.5006   |  1.448    |
|  25       | -9.606e-0 |  1.904    |  1.105    |
|  26       | -1.438e-0 |  1.648    |  2.508    |
|  27       | -1.421e-0 |  1.752    |  2.406    |
|  28       | -9.504e-0 |  1.91     |  1.151    |
|  29       | -1.429e-0 |  0.8126   |  1.159    |
|  30       | -1.454e-0 |  0.7834   |  1.946    |
|  31       | -1.491e-0 |  0.5398   |  1.98     |
|  32       | -1.219e-0 |  1.222    |  1.925    |
=================================================
```

```
|   iter    |  target   |   ATRsc   |   twind   |
-------------------------------------------------
|  1        | -6.628e-0 |  1.126    |  10.04    |
|  2        | -5.244e-0 |  0.5002   |  7.116    |
|  3        | -3.517e-0 |  1.981    |  5.003    |
|  4        | -6.346e-0 |  1.94     |  11.99    |
|  5        | -3.891e-0 |  0.5207   |  5.069    |
|  6        | -6.869e-0 |  1.767    |  11.99    |
|  7        | -3.751e-0 |  1.041    |  5.042    |
=================================================
```

```
|   iter    |  target   |   ATRsc   |   twind   |
-------------------------------------------------
|  1        | -3.832e-0 |  0.9868   |  5.642    |
|  2        | -4.551e-0 |  0.9891   |  6.144    |
|  3        | -3.095e-0 |  0.6128   |  4.002    |
|  4        | -4.488e-0 |  0.969    |  6.986    |
|  5        | -3.098e-0 |  0.6891   |  4.006    |
|  6        | -4.486e-0 |  0.6136   |  6.994    |
|  7        | -3.075e-0 |  0.6329   |  4.025    |
|  8        | -4.47e-05 |  0.6175   |  6.957    |
|  9        | -4.509e-0 |  0.9837   |  6.993    |
|  10       | -3.115e-0 |  0.9202   |  4.013    |
=================================================
```

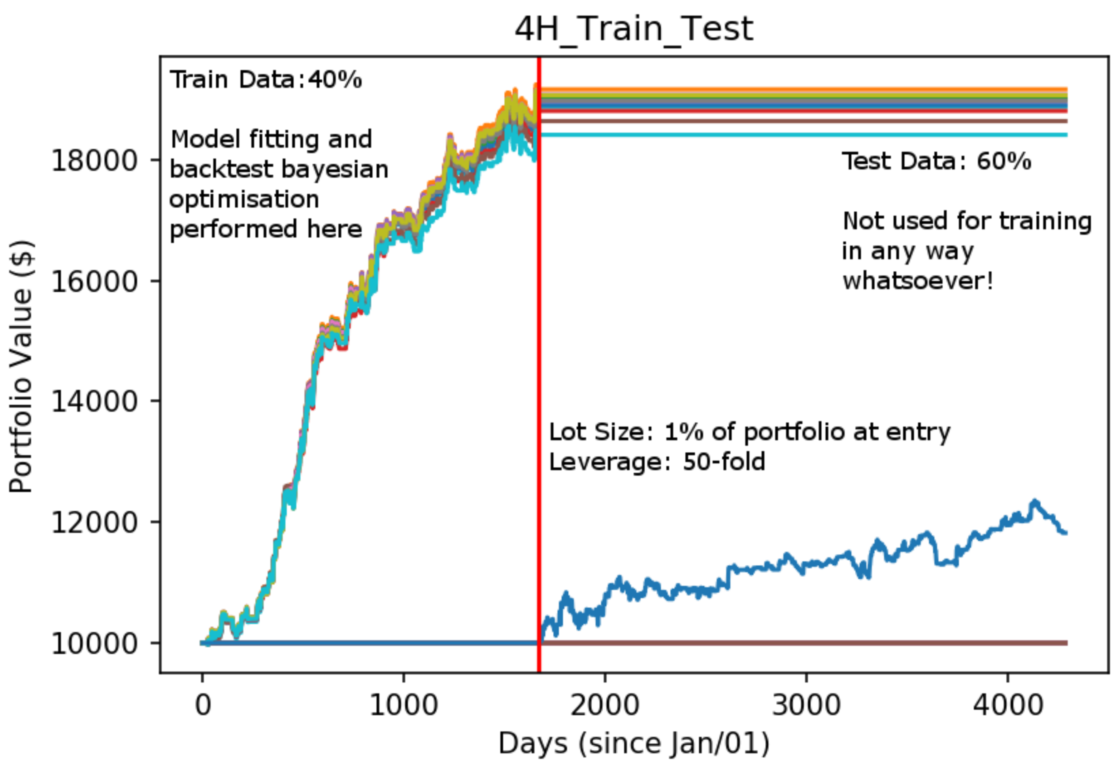
None of these work very well! I am ditching the trigger optimiser and going with some sensible handpicked values!

4H: Bayesian Optimisation of Trade Function -> Profit

```
optim_BackTest.maximize(init_points=4, n_iter=20)
|   iter    |  target   |   Qloss   |  Qprofit  | Risk_r... |
-------------------------------------------------------------
|  1        |  3.757    |  0.9382   |  0.868    |  1.119    |
|  2        |  8.042    |  1.262    |  0.9868   |  1.169    |
|  3        |  0.0      |  1.151    |  0.4432   |  1.034    |
|  4        |  0.01214  |  1.15     |  0.8075   |  1.302    |
|  5        |  0.0      |  1.084    |  0.4401   |  1.179    |
|  6        |  6.204    |  1.276    |  1.0      |  1.148    |
|  7        |  7.544    |  1.259    |  0.9973   |  1.179    |
|  8        |  3.717    |  1.271    |  0.9778   |  1.18     |
|  9        |  6.969    |  1.261    |  0.9954   |  1.167    |
|  10       |  6.782    |  1.257    |  0.9897   |  1.174    |
|  11       |  7.401    |  1.265    |  0.9914   |  1.172    |
|  12       |  6.495    |  1.264    |  0.9889   |  1.166    |
|  13       |  7.271    |  1.265    |  0.9899   |  1.178    |
|  14       |  7.386    |  1.262    |  0.996    |  1.175    |
|  15       |  7.168    |  1.264    |  0.9964   |  1.181    |
|  16       |  7.103    |  1.257    |  1.0      |  1.174    |
|  17       |  6.057    |  1.255    |  0.9819   |  1.173    |
|  18       |  6.527    |  1.262    |  0.9857   |  1.171    |
|  19       |  7.674    |  1.257    |  0.9912   |  1.166    |
|  20       |  7.094    |  1.261    |  0.993    |  1.179    |
|  21       |  5.076    |  1.248    |  0.9877   |  1.166    |
|  22         |  8.443      |  1.256      |  0.9958     |  1.181      |
|  23         |  7.951      |  1.269      |  0.9984     |  1.178      |
|  24         |  7.362      |  1.256      |  0.9962     |  1.187      |
```

Results

I spent a huge amount of time just figuring out which bits should or should not be optimised, which parameters to expose at which range etc. There are also a few erroneous bits of the timeseries data near the train/test split. I adjusted the split window a little bit so it would be excluded (otherwise the optimiser would just seek that out and exploit it at all costs).

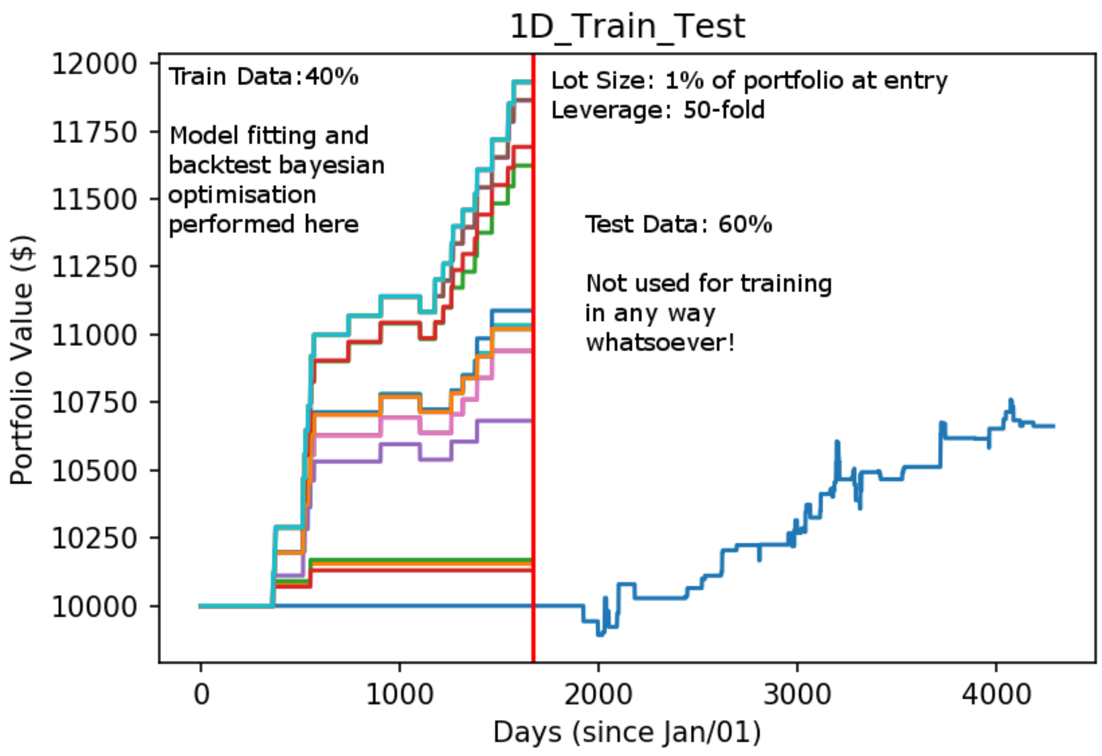


Result of model building on the 4H timeframe

Core parameters

- *ATR scaling factor: 0.8 (RF training),0.72 (System optimisation)*
- *Qloss: 0.915*
- *QProf: 0.93*
- *Risk_rate = 1.2*
- *Trade_window = 6 (6 * 4H = 24H!)*

This model will trade GBP/USD over the course of up to 24hrs based on the observation of a single 4hr move greater than 80% of the ATR20, in the EUR/USD pair.



Result of model building on the 4H timeframe

Core parameters

- *ATR scaling factor: 0.8 (RF training),0.79 (System optimisation)*
- *Qloss: 1.175*
- *QProf: 0.946*
- *Risk_rate = 1.2*
- *Trade_window = 1 (1D = 24H!)*

This model simply trades GBP/USD over a 24hr window based on a large move having occurred on the previous day in EUR/USD.

These two models worked well. Many models failed for many reasons, many of which were wonky optimisations. I found that the test models became profitable (& more robust to permutations) almost immediately after I made the decision to optimise less and fix the initial ATR scale at 0.8, fix the Risk-ratio at 1.2, and fix the time window at 24Hr for both. Simpler models work best!

Notes

Optimising the trigger function didn't really work. The issue was that minimising error by scale always pushed the time window to the minimum value, and ATR scale had minimal effect on it. The value of the scaling factor was best guessed at by hand.

Optimising the risk-ratio worked sometimes, but often led to problems. For example, the optimiser would often crank it up until nearly no trades were performed, then tweak the other params until profitable ones started popping above the threshold. Performing 4-5 1-day trades over 10 years didn't seem like a great idea, so this was scrapped too, 1.2 seems to work okay.

One-hour time-frame trading has yet to be attempted (not enough time right now to compute as the opt function takes 4x as long as the 4H).

The Bayesian optimiser is the definition of malicious compliance. If you give it a sword it will stab you with it.