UDACITY

‹ Return to Classroom

DISCUSS ON STUDENT HUB

Backtesting

REVIEW

CODE REVIEW

HISTORY

Meets Specifications

Congratulations on finishing the final project! 🎉 I'm sure the feeling of graduating from the nanodegree will be great. It is a great accomplishment! Along with applying for new roles if interested, you can also learn new skills from the Quant finance community like here https://www.worldquant.com/home/
You can also compete in machine learning competitions like on Kaggle www.kaggle.com
Good luck with your future endeavors 😄

Shift Daily Returns Data

The variable `frames` contains the data from `daily_return` and `data` correctly shifted.

There's a time delay of 2 in daily returns to make the live trading much more realistic. Check out this lesson video which describes how 2 day time-delay incorporating daily returns is realistic for simulated live trading.
https://www.youtube.com/watch?v=TYYV3MnhCP0

Build Universe Based on Filters

The function `get_universe` correctly creates a stock universe by selecting only those companies that have a market capitalization of at least 1 billion dollars (1e9) OR that are in the previous day's holdings, even if on the current day, the company no longer meets the 1 billion dollar criteria.

They should use the `.copy()` attribute to create a copy of the data and they should drop the column containing the daily return from the universe dataframe.

The returned `universe` dataframe should have a shape of (2265, 93)

Nice work in getting the universe of the stocks that will be used in forming the backtesting of the simulated live trading. The stocks in the previous day's holdings are to be considered because the trades are executed every day. It is good to drop the column containing the daily return from the universe dataframe. Because we need to make sure that we are not looking at returns when forming the portfolio. It is also stated in this rubric.

Factor covariance matrix

The function `diagonal_factor_cov` correctly creates the factor covariance matrix. The factor matrix must be scaled by (0.01**2). They must use the given `colnames` function to get the column names from `X` and use the statement 'covariance[date]' to get the covariances for the given `date`.

The returned factor covariance matrix should have shape (77, 77)

Alpha Combination

The function `get_B_alpha` correctly creates a matrix of alpha factors. They must use the given `get_formula` and `model_matrix` functions.

The returned `B_alpha` should be of type patsy.design_info.DesignMatrix and it should have shape (2265, 4). The 4 columns of this matrix should correspond to the 4 alpha factors chosen at the beginning, namely:

"USFASTD_1DREVRSL"
"USFASTD_EARNYILD"
"USFASTD_VALUE"
"USFASTD_SENTMT"

The function `get_alpha_vec` correctly creates a vector of alpha factors. To do this, they must add the rows of the Matrix of Alpha Factors and multiply the result by 1e-4.

The returned `alpha_vac` should have shape (2265,)

Objective function

The `obj_func(h)` function correctly implements the objective function. The equation of the objective function is given in the notebook.

Nice work in setting up the objective function to reduce the factor risk, idiosyncratic risk, and transaction costs and maximize the expected portfolio returns. You can relate this project 3 where you setup the portfolio optimization technique for the first time in AITND using `cvxpy`

Gradient

The `grad_func(h)` function correctly implements the gradient of the objective function. The equation of the gradient of the objective function is given in the notebook.

The optimizer which is used in the `scipy.optimize.fmin_l_bfgs_b` gets the closed formed solution. If you pass 1 in to the `approx_grad` as shown in the docs will make the optimizer to get the open formed solution.
https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.fmin_l_bfgs_b.html

Gradient descent gets the open-formed solution i.e. it is an iterative algorithm that takes smaller steps to the reach the optimal values. On the other hand, the optimizers that approach the problem analytically i.e. using the calculus/math that has been taught in the lessons.

Check out this lesson video https://www.youtube.com/watch?time_continue=1&v=rhVIF-nigrY
It is taken from extra-curricular section of *Deep Learning 24. Gradient Descent*

Now you can compare gradient descent with the closed form approach used within the `cvxpy` in this lesson video
https://www.youtube.com/watch?time_continue=118&v=ISRlP1GeOjU
Also check out this blog with 1. and 2. bullet points for quick comparison.
https://sebastianraschka.com/faq/docs/closed-form-vs-gd.html

Optimize

The function `get_h_star` correctly optimizes the objective function using the following functions `obj_func`, `grad_func`, and `scipy.optimize.fmin_l_bfgs_b`.

The returned `h_star` should have shape (2265,)

Risk Exposures

The function `get_risk_exposures` correctly calculates the portfolio's risk exposures

The returned `risk_exposures` Pandas series should have shape (77,). The index of this Pandas Series should correspond to the risk factors such as 'USFASTD_AERODEF', 'USFASTD_AIRLINES', 'USFASTD_ALUMSTEL', ...

The function `get_portfolio_alpha_exposure` correctly calculates the portfolio's alpha exposures.

The returned `portfolio_alpha_exposure` Pandas series should have shape (4,). The index of this Pandas Series should should correspond to the 4 alpha factors chosen at the beginning, namely:

"USFASTD_1DREVRSL"
"USFASTD_EARNYILD"
"USFASTD_VALUE"
"USFASTD_SENTMT"

Transaction Costs

The function `get_total_transaction_costs` correctly calculates the total transaction costs according to the equation given in the notebook.

You can always pickle and serialize these objects and save it to the workspace. So that you don't have to re-run all the variables again. That is you can pick up the work where you left off in the last session. Below are the code snippets of pickling the `port` dictionary which can also be done the same for the `trades`, `result` and `previous_holdings` variables

```
#saving the port file
import pickle
pickle.dump( port, open( "file-name.p", "wb" ) )
#loading the port file
import pickle
port = pickle.load( open( "file-name.p", "rb" ) )
```

Profit-and-Loss (PnL) attribution

Correctly calculate the PnL attributed to the alpha factors, the PnL attributed to the risk factors, and attribution to cost.

To calculate the alpha and risk exposures they must use the provided **`partial_dot_product`** function

Nice use of `partial_dot_product` function

Build portfolio characteristics

Correctly calculates the sum of long positions, short positions, net positions, gross market value, and amount of dollars traded.

DOWNLOAD PROJECT

RETURN TO PATH

Rate this review

START