In short, our goal is to replicate all the strategies written in OLPS Matlab.
To do this, we need to write our own code in python, and then compare the output plots with that in matlab to confirm our code is working.

Here are the step-by-step tutorial for you to get started with the replication:

**Preparation**

Step 1:
Clone OLPS matlab's github page.
https://github.com/OLPS/OLPS

Step 2:
Clone ETC's portfolio selection github page to your computer. (the following link is just an example using my github account, you would have to manually click into the folder)
https://github.com/polo2444172276/Survey_PortfolioSelection

Step 3:
All the necessary files for python replication are stored in 'OLPS_python' folder. These contain:

1) Datasets in the 'Datasets' folder
   These six dataset are exactly the same as those used in OLPS Matlab. I converted them into .xlsx format and added columns for each stock.
2) A replication tutorial pdf, which is what you're reading now.
3) A bunch of python modules.

There are two types of classes: Strategy and OLPSResult. Here Strategy class serves only as an abstract interface, and all strategy subclasses extend from it.
When you implement a strategy on a dataframe(more stricly, feed a dataframe object to a strategy object), an OLPSResult object will be returned, and its info can be plotted/printed.
But you don't need to understand the starting code in detail. Just getting a rough idea would be good to start the strategy replication.

**Replication**

The constant rebalanced portfolio(CRP) would be taken as an example for writing and testing the strategy.
Step 1:
Create a new module named 'CRP.py'. (In the starting code, all four benchmark strategies are included in the same module because they're short. But advanced strategies can be lengthy and should be in a single,separate module)
Step 2:
Write the necessary functions:
Make sure that *BCRP* extends from *Strategy*, and it has at least three functions: *__init__, run,* and *name.*
*__init__()* should take in hyperparameters that are determined before running the strategy. For example, learning rate, the number of nearest neighbours, threshold etc.
*name()* returns the full name of this strategy
*run()* takes in a pd.DataFrame object, and outputs an OLPS object.
As said previously, you don't need to know anything about OLPS objects. By following the related files in OLPS_Matlab -> Strategy, your goal is to generate a *np.array* named *rel_rets* that contains the relative return on each trading period, for example, [0.8,1.2,…] would mean 20% loss on first trading day, 20% gain on second, and so on.

And then, generate a time series and an OLPS object, by adding the following code as the last line of *run():*

*return OLPSResult.OLPS(pd.Series(index=df.index, data=rel_rets))*

```
#Constant Rebalanced Portfolio
class CRP(Strategy):
    def __init__(self, port:np.array):
        if not hp.isvalidport(port): raise Exception("Not a valid portfolio!")
        self.port = port
    def run(self, df:pd.DataFrame):
        #note that directly multiplying np arrays with "*" will produce incorrect results
        daily_rets = np.matmul(df.to_numpy(), self.port)
        ts_daily_rets = pd.Series(index=df.index, data=daily_rets.transpose())
        return OLPSResult(ts_daily_rets)
    def name(self):
        return "Constant Rebalanced Portfolio"
```

Make sure you add comments for clarification, generally 1 line of comment per 3-5 lines of code. You can also put helper functions wherever appropriate(inside/outside the class).

Step 4:

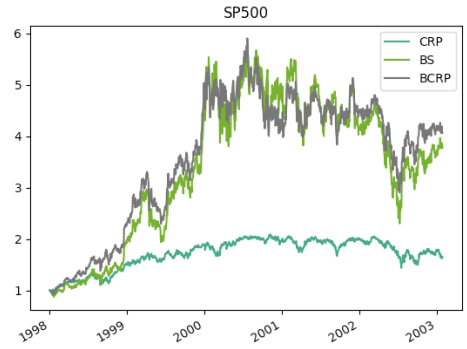
Edit the following code in the *main()* interface of OLPSrun.py:

```
if __name__ == "__main__":
    # load data
    # df_nyseo = pd.read_excel("Datasets/nyse-o.xlsx")
    # df_nysen = pd.read_excel("Datasets/nyse-n.xlsx")
    # df_tse = pd.read_excel("Datasets/tse.xlsx")
    df_sp500 = pd.read_excel("Datasets/sp500.xlsx")
    # df_msci = pd.read_excel("Datasets/msci.xlsx")
    # df_djia = pd.read_excel("Datasets/djia.xlsx")

    # #add time column as index
    # add_time(df_nyseo, "1962-07-03", "1984-12-31")
    # add_time(df_nysen, "1985-01-01", "2010-06-30")
    # add_time(df_tse, "1994-01-04", "1998-12-31")
    add_time(df_sp500, "1998-01-02", "2003-01-31")
    # add_time(df_msci, "2006-04-01", "2010-03-31")
    # add_time(df_djia, "2001-01-14", "2003-01-14")


    df_name = "SP500"
    #uniformly distribute wealth on all stocks
    num_stocks = df_sp500.shape[1]
    portfolio = [1/df_sp500.shape[1] for i in range(num_stocks)]
    #a list of all strategies to be tested
    strats = [ bms.CRP(portfolio), bms.BS(), bms.BCRP()]
    #compare them on S&P500
    compare_strats(strats, df_sp500, df_name=df_name, print_option=True,plot_option=True )
```

Comment/Uncomment the corresponding dataset you would like to use.
Define necessary input variables.
Change *df_name* into an appropriate string.
Create and add the desired strategy objects in *strats.*

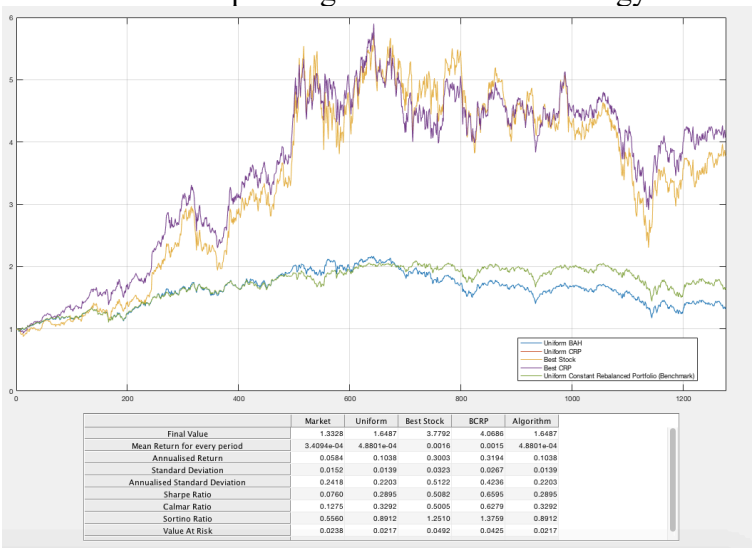
Step5:

Run OLPSrun.py. It outputs the following graph and prints info:




Step6:
Run OLPS_gui.m.
To do this, make sure you are in the folder OLPS_Matlab.


Then type in OLPS_gui in command window.
A window will pop up. Choose Algorithm Analyzer.
Choose the corresponding dataframes and strategy and click 'START'.




| | Market | Uniform | Best Stock | BCRP | Algorithm |
|---|---|---|---|---|---|
| Final Value | 1.3328 | 1.6487 | 3.7792 | 4.0686 | 1.6487 |
| Mean Return for every period | 3.4094e-04 | 4.8801e-04 | 0.0016 | 0.0015 | 4.8801e-04 |
| Annualised Return | 0.0584 | 0.1038 | 0.3003 | 0.3194 | 0.1038 |
| Standard Deviation | 0.0152 | 0.0139 | 0.0323 | 0.0267 | 0.0139 |
| Annualised Standard Deviation | 0.2418 | 0.2203 | 0.5122 | 0.4236 | 0.2203 |
| Sharpe Ratio | 0.0760 | 0.2895 | 0.5082 | 0.6595 | 0.2895 |
| Calmar Ratio | 0.1275 | 0.3292 | 0.5005 | 0.6279 | 0.3292 |
| Sortino Ratio | 0.5560 | 0.8912 | 1.2510 | 1.3759 | 0.8912 |
| Value At Risk | 0.0238 | 0.0217 | 0.0492 | 0.0425 | 0.0217 |

Step 7:
Compare the results of python and matlab.
As you can see, the plot in python match with the corresponding lines in Matlab output. This is a sign of successful replication! The details(annualized return, Sharpe Ratio..) can be verified too.

Some of you were worried about your coding skills for this project. There is no need to. However, if you haven't, you still need to learn to use pandas, numpy, scipy and possibly other packages to complete the replication task. Their usage are listed as follows:
Pandas:  to deal with dataframes(tables, .csv, time series)
Numpy: linear algebra
Scipy: other mathematical tools such as optimization

They should be relatively straightforward to implement. Google should serve as the biggest helper. Feel free to ask questions though.

We will determine allocation of the strategies in the weekly meeting.
If you have any questions regarding anything, feel free to pin me.
Thank you for reading this! ☺.

Polo