CS7646 ML4T
Machine Learning
for Trading

PROJECT 2: OPTIMIZE SOMETHING

REVISIONS

This assignment is subject to change up until 3 weeks prior to the due date. We do not anticipate changes; any changes will be logged in this section.

OVERVIEW

In this project, you will write software that evaluates and prepares portfolio metrics. You will submit the code for the project in Gradescope SUBMISSION. You will also submit to Canvas a chart as a 1-page report that compares two normalized portfolios.

ABOUT THE PROJECT

Revise the optimization.py code to return several portfolio statistics: stock allocations (allocs), cumulative return (cr), average daily return (adr), standard deviation of daily returns (sddr), and Sharpe ratio (sr). This project builds upon what you learned about portfolio performance metrics and optimizers to optimize a portfolio. That means that you will find how much of a portfolio's funds should be allocated to each stock to optimize its performance. While a portfolio can be optimized for many different metrics, in this version of the assignment we will maximize the Sharpe Ratio. You will use the data cleansing techniques introduced in the course lectures.

YOUR IMPLEMENTATION

Implement the optimize_portfolio() function

The optimization.py file must implement this API specification (including the default arguments).

Revise the code to implement a Python function named optimize_portfolio() in the file optimization.py that can find the optimal allocations for a given set of stocks. You should optimize for maximum Sharpe Ratio taking long positions only.

The function should accept as input a list of symbols as well as start and end dates and return a list of floats as a one-dimensional Numpy array that represents the allocations to each of the equities. You can take advantage of routines developed in the optional assess portfolio (*see note under Starter Code*) project to compute daily portfolio value and statistics, and then by using cut-and-paste to move the code from the project for the necessary functions into optimization.py. Otherwise, you will need to create the functions within optimization.py.

You are given the following inputs for optimizing a portfolio:

- A date range to select the historical data to use (specified by a start and end date)
- Symbols for equities (e.g., GOOG, AAPL, GLD, XOM). Note: You should support any symbol in the data directory. Your code should support any number of assets >= 2. Don't hardcode 4 as the number of assets.

Your goal is to find allocations to the symbols that optimize the criterion given above. Assume 252 trading days in a year and a risk-free return of 0.0 per day.

You will produce a single chart, as a .png file called "Figure1.png", comparing the optimal normalized portfolio with SPY using the following portfolio parameters:

- Start Date: 2008-06-01, End Date: 2009-06-01, Symbols: ['IBM', 'X', 'GLD', 'JPM'].

While four (4) symbols are used in the portfolio associated with Figure 1, the implementation must be robust enough to handle any number of symbols.

The chart must be correct, be properly titled, have appropriate axis labels, use consistent axis ranges, and have legends. You should use python's Matplotlib library.

The implementation will be evaluated using 10 test cases that will validate that the sum of the allocations to 1.0 (with a -/+ 0.2 tolerance), the individual allocations range from 0.0 to 1.0 (with a -/+ 0.2 tolerance), and that the standard deviation is correct (within 5% of the reference implementation).

Technical Requirements

Use only the functions provided in the util.py file for reading historical stock data provided in ../data folder. Do NOT modify, copy, or move the util.py file or its functions.

Your code must run in less than 5 seconds per test case in the university-provided Gradescope environment.

A random seed may not be set for this assignment.

Watermarked Charts (i.e., where the GT Username appears over the lines) can be shared in the designated pinned (e.g., “Project 2- Student Charts”) thread alone. Charts presented in reports or submitted for grading must not contain watermarks.

Example

The following example illustrates how the optimize_portfolio function will be called:

```
import datetime as dt
allocs, cr, adr, sddr, sr =
 optimize_portfolio(sd=dt.datetime(2008,1,1), ed=dt.datetime(2009,1,1),
 syms=['GOOG','AAPL','GLD','XOM'], gen_plot=False)
```

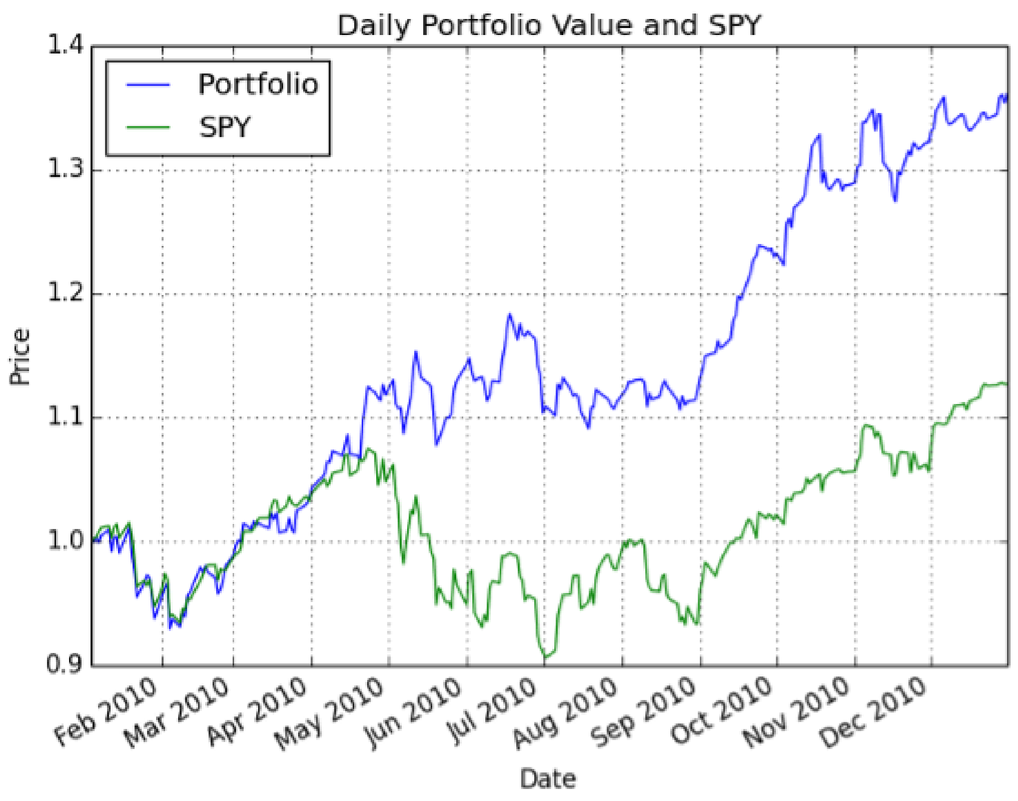
Where the returned output is:

- allocs: A 1-d NumPy NDArray of allocations to the stocks. All the allocations must be between 0.0 and 1.0 and they must sum to 1.0. Some of the allocations may be 0.
- cr: Cumulative return
- adr: Average daily return
- sddr: Standard deviation of daily return
- sr: Sharpe ratio

The input parameters are:

- sd: A DateTime object that represents the start date
- ed: A DateTime object that represents the end date
- syms: A list of symbols that make up the portfolio (note that your code should support any symbol in the data directory)
- gen_plot: If True, optionally create a plot named plot.png. This may be useful for any tests you develop where you want to print output to the console. The Autograder will always call your code with gen_plot = False.

An example chart that you might create for assessing your optimizer.



Suggestions

- Refer to comments in the provided helper code for pointers on implementation. In order to specify bounds and constraints when using the scipy.optmize module, you'll need to use a special syntax explained here: http://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html
- For bounds, you simply need to pass in a sequence of 2-tuples (<low>, <high>). Just remember that you need to supply as many tuples as the number of stocks in your portfolio.
- For constraints, it's a little tricky. You need to pass in a sequence of dicts (dictionaries), one dictionary per constraint. Each dictionary must specify the type of constraint ('eq' for equality, or 'ineq' for inequality), and a function that returns 0 only when the input satisfies the constraint (this is the same input that is supplied to your evaluation function). E.g. to constrain the sum of all values in the input array to be less than 50, you could pass in the following (lambdas are just anonymous functions defined on the spot):

constraints = ({ 'type': 'ineq', 'fun': lambda inputs: 50.0 – np.sum(inputs) })

- Use a uniform allocation of 1/n to each of the n assets as your initial guess.

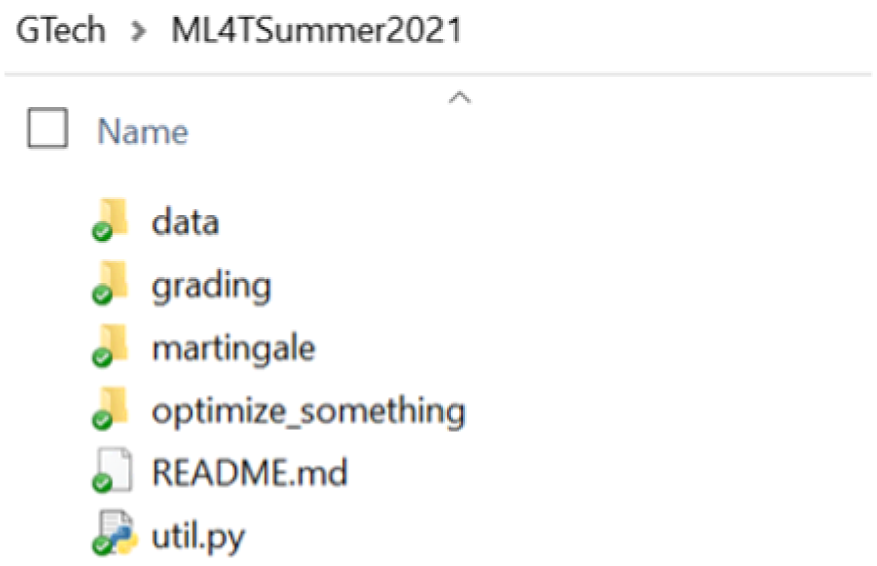
TECHNICAL REQUIREMENTS

The optimization.py file must implement this API specification.

Starter Code

To make it easier to get started on the project and focus on the concepts involved, you will be given a starter framework. This framework assumes you have already set up the local environment and ML4T Software. The framework for Project 1 can be obtained from: Optimize_Something2021Summer.zip.

Extract its contents into the base directory (e.g., ML4T_2021Summer). This will add a new folder called "optimize_something" to the directory structure.



Within the optimize_something folder are two files: optimization.py, and grade_optimization.py. You will modify the optimization.py file to implement the necessary functionality for this assignment. You must implement the optimize_portfolio() function. The existing code in the optimization.py file is best thought of as "stub" or "starter" code that may contain ideas for functions and methods that might be used in your implementation. This file must remain and run from within the optimize_something directory using the following command:

```
PYTHONPATH=../:. python optimization.py
```

Note: You can leverage the functions in the now deprecated (no longer assigned) assess portfolio project that assessed the value of a portfolio with a given set of allocations. This is given only as a potential tool to use while completing the project to ensure you have your portfolio statistic calculations correct. Note that this code was writing for Python 2.7 and may need to be updated to run with Python 3.6.

CONTENTS OF REPORT

The report.pdf file will consist of a single page that contains the chart generated by your implementation. The charts must be generated by the code and saved to a file. It can be imported (without additional post-processing) into the report document. The chart will be evaluated on its correctness (i.e., shape, scale, range, normalizes lines, current number of lines) and textual completeness (i.e., appropriate title, legends, axis labels). All textual elements must be readable and non-overlapping.

TESTING

To test your code, we will call the optimize_portfolio() function only. You are encouraged to perform any tests necessary to build confidence that the code will run properly when submitted for grading and will produce the required results. To run and test that the file will run from within the optimize_something directory, use the command given above at the command prompt from within the optimize_something directory:

```
PYTHONPATH=../:. python optimization.py
```

Note: Once submitted for grading, we will use the above command to call the "__main__" section only. The program should run in its entirety and produce the necessary output and chart.

Additionally, we provide the grade_optimization.py file that can be used for your tests. This file is the same script that will be run when the code is submitted to GradeScope TESTING. This file is not a complete test suite and does not match the more stringent private grader that is used in Gradescope SUBMISSION. To run and test that the file will run from within the optimize_something directory, use the command:

```
PYTHONPATH=../:. python grade_optimization.py
```

You are encouraged to submit your files to Gradescope TESTING, where some basic pre-validation tests will be performed against the code. Gradescope TESTING does not grade your assignment. No credit will be given for coding assignment that do not pass this pre-validation.

You are allowed **unlimited** resubmissions to Gradescope **TESTING**. Please refer to the Gradescope Instructions for more information.

SUBMISSION INSTRUCTIONS

This is an individual assignment. All work you submit should be your own. Make sure to cite any sources you reference and use quotes and in-line citations to mark any direct quotes.

Late work is not accepted without advanced agreement except in cases of medical or family emergencies. In the case of such an emergency, please contact the Dean of Students.

Report Submission

Complete your assignment using the JDF format, then save your submission as a PDF. The report is to be submitted as **report.pdf.** None of the JDF sections identified in the template are required for this assignment. Assignments should be submitted to the corresponding assignment submission page in Canvas. You should submit a single PDF for this assignment. Please submit the following file to Canvas in PDF format only:

- **report.pdf**

Do not submit any other files. The chart must be included in the report, not submitted as a separate file. Also note that when we run your submitted code, it should generate the chart. Not submitting a report will result in a penalty.

You are allowed unlimited submissions of the report.pdf file to Canvas.

Code Submission

This class uses Gradescope, a server-side auto-grader, to evaluate your code submission. No credit will be given for code that does not run in this environment and students are encouraged to leverage Gradescope TESTING prior to submitting an assignment for grading.

Please submit the following file to Gradescope SUBMISSION:

- **optimization.py**

Do not submit any other files.

You are allowed a MAXIMUM of three (3) code submissions to Gradescope SUBMISSION.

GRADING INFORMATION

Your report is worth 20% of your grade. As such, it will be graded on a 20-point scale coinciding with a rubric design to mirror the questions above. Make sure to answer those questions. The submitted code (which is worth 80% of your grade) is run as a batch job after the project deadline. The code will be graded using a rubric design to mirror the implantation details above, where each of the 10 implementation test cases is worth 8 points. Deductions will be applied for unmet implementation requirements or code that fails to run.

We do not provide an explicit set timeline for returning grades, except that everything will be graded before the institute deadline (end of the term). As will be the case throughout the term, the grading team will work as quickly as possible to provide project feedback and grades. Once grades are released, any grade related matters must follow the Assignment Follow-Up guidelines and process.

RUBRIC

Report [20 points]

Each chart line (SPY/Portfolio is worth 10 points each)

- No chart or chart is total nonsense. (-20 points, if incorrect).
- Chart is wrong shape (e.g., wrong time period) (-10 point if incorrect).
- Partly wrong shape (chart begins correctly but midway or after is incorrect) (-5 points if incorrect).
- Chart not normalized or -5 chart data scale substantially wrong (e.g., right shape that is normalized to 1 but the max/end values are wrong – for example, 3.5 instead of 2.5). (-10 points if incorrect).
- Missing required data series (did not plot one of portfolio or SPY lines) (-10 points per case, if incorrect).
- Chart labels/text/legend unreadable (e.g., too small, labels overlap a lot) or missing. (-2 points per each case, if missing or incorrect).
- Min score: 0.

Code

- Does the code generate the appropriate chart written to a .png file? (-10 points).
- Does the code create charts that are displayed on the screen or in a window when the code is run? (-10 points for each chart).

Auto-Grader [80 points]

We will test your code against 10 cases (8 points per case). Each case will be deemed “correct” if:

- sum(allocations) = 1.0 +- 0.02 (2 points)
- Each allocation is between 0 and 1.0 +- 0.02 (negative allocations are allowed if they are very small) (2 points)
- Standard deviation of daily return of the allocated portfolio is within 5% of reference solution or lower (4 points)
- Minimum score: 0.

There is no partial credit for the per-test-case points breakdown above if outside the threshold. If you are within the threshold, you get the point allocation, outside the threshold you get 0.

Allowed

- All libraries provided in the Local Environment setup page.
- All standard Python libraries (except the os library, which is prohibited)
- Code provided by the instructor or allowed by the instructor to be shared.

Prohibited

- Use of the Python os library/module.
- Use of global variables.
- Code that results in a window or chart being displayed (an example of which is matplotlib’s plt.show() function)
- Code that creates any directories
- Any code that you did not write yourself (except for properly cited code provided by the instructor).

- Absolute import statements, such as: *from folder_name import martingale*.
- Any libraries not included as part of the Local Environment as provided on the Local Environment Setup page.

- Editing (i.e., adding, changing or deleting) above the line that reads: "—–do not edit anything above this line—".
- Sharing of tables and statistics.
- Camels and other dromedaries.

RELEVANT RESOURCES

You may find the following resources useful in completing the project or in providing an in-depth discussion of the material.

Sharpe (1994), The Sharpe Ratio

(Deprecated) Assess Portfolio Files

SciPy.org, scipy.optimize.minimize

TutorialsPoint.com, What are default arguments in Python?

Hilpisch (2014 – 1st Edition or 2018 – 2nd Edition), Python for Finance (Chapter 9 – Mathematical Tools)

Martelli, A. Ravenscroft, and S. Holden (2017), Python in a Nutshell, 3rd Edition