Fall 2019 Project 8: Strategy Learner

From Quantitative Analysis Software Courses

Contents

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 design a learning trading agent. You must draw on the learners you have created so far in the course. Your choices are:

1. **Regression** or **classification**-based learner: Create a strategy using your Random Forest learner. Suggestions if you follow this approach: Classification_Trader_Hints. Important note, if you choose this method, you must set the leaf_size for your learner to 5 or greater. This is to avoid degenerate overfitting in-sample. For classification, you must use mode rather than mean (RTLearner, BagLearner).
2. **Reinforcement Learner**-based approach: Create a Q-learning-based strategy using your Q-Learner. Read the Classification_Trader_Hints first, because many of the ideas there are relevant for the Q trader, then see Q_Trader_Hints For Q-learning, use the same binning cuts for in-sample and out-of-sample.
3. **Optimization**-based learner: Create a scan-based strategy using an optimizer. Read the Classification_Trader_Hints first, because many of the ideas there are relevant for the Opto trader, then see Opto_Trader_Hints

Regardless of your choice above, your learner should work in the following way:

- In the training phase (e.g., addEvidence()) your learner will be provided with a stock symbol and a time period. It should use this data to learn a strategy. For instance, for a regression-based learner it will use this data to make predictions about future price changes.
- In the testing phase (e.g., testPolicy()) your learner will be provided a symbol and a date range. All learning should be turned OFF during this phase.
- You should use exactly the same indicators as in the manual strategy project so we can compare your results. You may optimize your indicators for time (vectorization), but there should be no changes to lookback

windows or any other pertinent parameters.

If the date range is the same as used for the training, it is an in-sample test. Otherwise it is an out-of-sample test. Your learner should return a trades dataframe like it did in the last project. Here are some important requirements: Your testPolicy() method should be much faster than your addEvidence() method. The timeout requirements (see rubric) will be set accordingly. Multiple calls to your testPolicy() method should return exactly the same result.

Overall, your tasks for this project include:

- Devise numerical/technical indicators to evaluate the state of a stock on each day.
- Build a strategy learner based on one of the learners described above that uses the indicators.
- Test/debug the strategy learner on specific symbol/time period problems.
- Write a report describing your learning strategy.

Scoring for the project will be based on trading strategy test cases and a report.

Template

Instructions:

- Download and install the files from this zip file File:19fall strategy learner.zip
- Place your existing Q-Learner or RTLearner (Okay to include DTLearner as well if inheritance is involved) or OptimizationLearner into the `strategy_learner/` directory.
- Implement the `StrategyLearner` class in `strategy_learner/StrategyLearner.py`
- See "what to turn in" below for a list of files that should be submitted.
  - Your learning code: `QLearner.py`, `RTLearner.py`, `BagLearner.py`, and/or `OptimizeLearner.py`
- To test your strategy learner, follow the instructions on Running the grading scripts

Data Details, Dates and Rules

- Use only the data provided for this course. You are not allowed to import external data.
- For your report, trade only the symbol JPM. This will enable us to more easily compare results. We will test your learner with other symbols as well.
- You may use data from other symbols (such as SPY) to inform your strategy.
- The in sample/development period is January 1, 2008 to December 31 2009.
- The out of sample/testing period is January 1, 2010 to December 31 2011.
- Starting cash is $100,000.
- Allowable positions are: 1000 shares long, 1000 shares short, 0 shares.
- Benchmark: The performance of a portfolio starting with $100,000 cash, investing in 1000 shares of the symbol in use and holding that position. Include transaction costs.
- There is no limit on leverage.
- Transaction costs: Commission will always be $0.00, Impact may vary, and will be passed in as a parameter to the learner.
- Minimize use of herrings.

Implement Strategy Learner

For this part of the project you should develop a learner that can learn a trading policy using your learner. You should be able to use your Q-Learner or RTLearner from the earlier project directly. If you want to use the optimization approach, you will need to create new code or that. You will need to write code in

`StrategyLearner.py` to "wrap" your learner appropriately to frame the trading problem for it. Utilize the template provided in `StrategyLearner.py`. Be sure to add in an author method.

Your StrategyLearner should implement the following API:

```
import StrategyLearner as sl
learner = sl.StrategyLearner(verbose = False, impact = 0.000) # constructor
learner.addEvidence(symbol = "AAPL", sd=dt.datetime(2008,1,1), ed=dt.datetime(2009,12,31), sv = 100000) # training
df_trades = learner.testPolicy(symbol = "AAPL", sd=dt.datetime(2010,1,1), ed=dt.datetime(2011,12,31), sv = 100000)
```

The input parameters are:

- verbose: if False do not generate any output
- impact: The market impact of each transaction.
- symbol: the stock symbol to train on
- sd: A datetime object that represents the start date
- ed: A datetime object that represents the end date
- sv: Start value of the portfolio

The output result is:

- df_trades: A data frame whose values represent trades for each day. Legal values are +1000.0 indicating a BUY of 1000 shares, -1000.0 indicating a SELL of 1000 shares, and 0.0 indicating NOTHING. Values of +2000 and -2000 for trades are also legal when switching from long to short or short to long so long as net holdings are constrained to -1000, 0, and 1000.

Implement author() function (deduction if not implemented)

You should implement a function called `author()` that returns your Georgia Tech user ID as a string. This is the ID you use to log into Canvas. It is not your 9 digit student number. Here is an example of how you might implement author():

```
    def author():
        return 'tb34' # replace tb34 with your Georgia Tech username.
```

Implementing this method correctly does not provide any points, but there will be a penalty for not implementing it.

Contents of Report

Write a report describing your system. The centerpiece of your report should be the description of how you utilized your learner to determine trades:

- Describe the steps you took to frame the trading problem as a learning problem for your learner. What are your indicators? (They should be the same ones used for Manual Strategy assignment) Describe how you discretized (standardized) or otherwise adjusted your data. If not, tell us why not.

- Experiment 1: Using exactly the same indicators that you used in manual_strategy (trade JPM), compare your manual strategy with your learning strategy in sample. You can use the same impact (.005) as was used for Project 6 or use 0 for both. Be sure to add in an author method.
  - Describe your experiment in detail: Assumptions, parameter values and so on.

  - Describe the outcome of your experiment.
  - Would you expect this relative result every time with in-sample data? Explain why or why not.
- Experiment 2: Provide a hypothesis regarding how changing the value of `impact` should affect in sample trading behavior and results (provide at least two metrics). Conduct an experiment with JPM on the in sample period to test that hypothesis. Provide charts, graphs or tables that illustrate the results of your experiment. The code that implements this experiment and generates the relevant charts and data should be submitted as `experiment2.py`. Be sure to add in an author method.

Your descriptions should be stated clearly enough that an informed reader could reproduce the results without referencing your code.

The report can be up to 2500 words long and contain up to 6 figures (charts and/or tables).

What to turn in

Turn your project in via canvas. **All submitted code must include comments with your name and User ID.**

- Your learner.
- Your StrategyLearner as `StrategyLearner.py`
- Your report as `report.pdf`.
  - Please submit the report (.pdf) into the assignment "Project 8: Strategy Learner (Report)" only

- ALL of your code should be contained in the files:
  - `indicators.py` Your code that implements your indicators.
  - `marketsimcode.py` Optional if needed to support your strategy learner. An improved version of your marketsim code that accepts a "trades" data frame (instead of a file).
  - `StrategyLearner.py`
  - `ManualStrategy.py`
  - `experiment1.py`
  - `experiment2.py`
  - Your learning code: QLearner.py, or RTLearner (okay to submit DTLearner if using inheritance) and BagLearner.py or OptimizeLearner.py.
- Make sure that you submit all code that is necessary for your software to run including plotting. If your code crashes because of a missing file, you may lose signficant points on the code section.
- Do not submit any other files. Penalties will be assessed for additional files outside of the allowed ones as stated above.
- Please submit the code (.py) into the assignment "Project 8: Strategy Learner (Code)" only

Rubric

Code: 70 points

We will test StrategyLearner in the following situations:

- Training / in sample: January 1, 2008 to December 31 2009.
- Testing / out of sample: January 1, 2010 to December 31 2011.
- Symbols: ML4T-220, AAPL, UNH, SINE_FAST_NOISE
- Starting value: $100,000
- Benchmark: Buy 1000 shares on the first trading day, Sell 1000 shares on the last day.
- Commissions = $0.00, impact = 0.00

We expect the following outcomes in evaluating your system:

- For ML4T-220
  - addEvidence() completes without crashing within 25 seconds: 1 points
  - testPolicy() completes in-sample within 5 seconds: 2 points
  - testPolicy() returns same result when called in-sample twice: 2 points
  - testPolicy() returns an in-sample result with cumulative return greater than 100%: 5 points
  - testPolicy() returns an out-of-sample result with cumulative return greater than 100%: 5 points
- For AAPL
  - addEvidence() completes without crashing within 25 seconds: 1 points
  - testPolicy() completes in-sample within 5 seconds: 2 points
  - testPolicy() returns same result when called in-sample twice: 2 points
  - testPolicy() returns an in-sample result with cumulative return greater than benchmark: 5 points
  - testPolicy() returns an out-of-sample result within 5 seconds: 5 points
- For SINE_FAST_NOISE
  - addEvidence() completes without crashing within 25 seconds: 1 points
  - testPolicy() completes in-sample within 5 seconds: 2 points
  - testPolicy() returns same result when called in-sample twice: 2 points
  - testPolicy() returns an in-sample result with cumulative return greater than 200%: 5 points
  - testPolicy() returns an out-of-sample result within 5 seconds: 5 points
- For UNH
  - addEvidence() completes without crashing within 25 seconds: 1 points
  - testPolicy() completes in-sample within 5 seconds: 2 points
  - testPolicy() returns same result when called in-sample twice: 2 points
  - testPolicy() returns an in-sample result with cumulative return greater than benchmark: 5 points
  - testPolicy() returns an out-of-sample result within 5 seconds: 5 points
- Witheld test case 1: In sample test case for an unknown symbol.
  - If any part of code crashes: 0 points awarded.
  - testPolicy() returns an in-sample result with cumulative return greater than benchmark: 5 points
- Withheld test case 2: In sample test case to verify that strategy accounts for different values of `impact`
  - If any part of code crashes: 0 points awarded.
  - Learner returns different trades when impact value is significantly different: 5 points
- Is any submitted code lacking a comment with the student's name and User ID? -5 points for each occurance, up to a maximum of -20
- Does the submitted code and report reflect an understanding of the subject matter? (up to -50 if not)

We reserve the right to use different time periods if necessary to reduce auto grading time.

- IMPORTANT NOTES
  - For achieving the required cumulative return, recall that `cr = (portval[-1]/portval[0]) - 1.0`
  - The requirement that consecutive calls to `testPolicy()` produce the same output for the same input means that you **cannot** update, train, or tune your learner in this method. For example, a solution that uses Q-Learning should use `querySetState()` and **not** `query()` in `testPolicy()`. Updating, training, and tuning (`query()`) is fine inside `addEvidence()`.
  - Your learner should **not** select different hyper-parameters based on the **symbol**. Hyper-parameters include (but are not limited to) things like features, discretization size, sub-learning methods (for ensemble learners). Tuning using cross-validation or otherwise pre-processing the **data** is OK, things like `if symbol=="UNH"` are **not OK**. There will be a withheld test case that checks your code on a valid symbol that is not one of the four listed above.
  - Presence of code like `if symbol=="UNH"` will result in a 20 point penalty.
  - When evaluating the trades generated by your learner, we **will** consider transaction costs (market impact).

Report: 30 points

- Does the submitted code `indicators.py` properly reflect the indicators provided in the report (-20 points if not)
- Are the indicators used for Strategy Learner the same as the ones used in Manual Strategy (up to -20 point if not)
- Is the method by which the learner is utilized to create a trading strategy described sufficiently clearly that an informed reader could reproduce the results without referencing your code? (up to 10 point deduction if not)
- Does report description match the code? (up to 10 point deduction if not)
- Are the two required experiments explained well? (up to 5 points deduction each if not)
- Are the two required experiments compellingly supported with tabular or graphical data? (up to 5 points deduction each if not)
- Does the report contain more than 2500 words? (10 point deduction if so)
- Does the report contain more than 6 figures and/or tables? (10 point deduction if so)
- Does the submitted code and report reflect an understanding of the subject matter and follow the assignment directions? (up to -20 points if not)
- Missing author method. (up to -5 for each missing with max 10 deduction for assignment)
- Is the report especially well written (up to 2 point bonus)
- Did the student choose to use optimization based learning, and did their learning strategy beat the manual strategy (+1 point bonus)
- Does the submitted code produce all included charts, with one run and within the time limit without any manipulation? **DO NOT use plt.show() and manually save your charts. The charts should be created and saved using Python code** (-10 points each up to a max of -20 if not) (up to -20 points if not)

Required, Allowed & Prohibited

Required:

- Your project must be coded in Python 3.6.x.
- Your code must run on one of the university-provided computers (e.g. buffet02.cc.gatech.edu).
- All code must be your own.
- All charts must be created within Python and the code used to create them must be submitted. It should generate ALL plots at one run, without ANY code manipulation on the TAs part. It should also run within 5 minutes.
- No external learning libraries allowed.

Allowed:

- You can develop your code on your personal machine, but it must also run successfully on one of the university provided machines or virtual images.
- Your code may use standard Python libraries.
- You may use the NumPy, SciPy, matplotlib and Pandas libraries. Be sure you are using the correct versions.
- Code provided by the instructor, or allowed by the instructor to be shared.
- Sardines.
- Use util.py (only) for reading data. Only use the API methods provided here to read in stock data. Do NOT modify this file or copy the code from it into other files. For grading, we will use our own unmodified version.

Prohibited:

- Any libraries not listed in the "allowed" section above.
- Any code you did not write yourself.
- Any Classes (other than Random) that create their own instance variables for later use (e.g., learners like kdtree).
- Print statements outside "verbose" checks (they significantly slow down auto grading).
- Any method for reading data besides util.py

- Excessive use of herrings.

Legacy

- MC3-Project-4-Legacy-Q-trader
- MC3-Project-2-Legacy-trader
- MC3-Project-2-Legacy
- 2018-Strategy-Learner-Legacy
