MC3-Project-4

From Quantitative Analysis Software Courses

Contents

Updates / FAQs

- **2017-7-11**
  - Project revision in progress.
- **2017-7-17**
  - Project finalized.

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:

- Create a regression or classification-based 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.
- 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
- Create a scan-based strategy using an optimizer.

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 its 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.

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 set of dated trades similar to those input to your market simulator.

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 and Data

- Update your local repository from github.
- Place your existing Q-Learner or RTLearner or OptimizationLearner into `mc3p4_strategy_learner/`.
- Implement the `StrategyLearner` class in `mc3p4_strategy_learner/StrategyLearner.py`
- ALL of your code should be contained in the two files listed above.
- To test your strategy learner, follow the instructions on Running the grading scripts

Use the following parameters for trading and evaluation:

- Use only the data provided for this course. You are not allowed to import external data.
- Allowable positions are: 200 shares long, 200 shares short, 0 shares.
- Benchmark:
  - The performance of a portfolio starting with $100,000 cash, then investing in 200 shares of the relevant symbol and holding that position
- There is no limit on leverage.
- Use the transaction cost model from MC2-Project-1 when evaluating your portfolio. (IE: commissions and market impact)

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, with no changes. 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`.

Your StrategyLearner should implement the following API:

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

The input parameters are:

- verbose: if False do not generate any output
- 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 +200.0 indicating a BUY of 200 shares, -200.0 indicating a SELL of 200 shares, and 0.0 indicating NOTHING. Values of +400 and -400 for trades are also legal so long as net holdings are constrained to -200, 0, and 200.

Contents of Report

Write a report describing your system. The centerpiece of your report should be the description of how you have utilized your learner to determine trades. Describe the steps you took to frame the trading problem as a learning problem for your learner.

In the course of creating your learning trading strategy you will probably evaluate a number of different (hyper-)parameters for your learner and your trading strategy. Choose two of those to look at more carefully. Conduct and report on two experiments that illustrate the methods by which you refined your learner or strategy to excel at the assigned task.

Your descriptions should be stated clearly enough that an informed reader could reproduce the results you report.

The report can be up to 2000 words long and contain up to 6 figures (charts and/or tables).

What to turn in

Turn your project in via t-square. Your submission should include exactly 3 files. All of your code must be contained within two files: your learner and StrategyLearner.py.

- Your learner.
- Your StrategyLearner as `StrategyLearner.py`
- Your report as `report.pdf`
- Do not submit any other files.

Rubric

Code: 65 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 200 shares on the first trading day, Sell 200 shares on the last day.

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
- For withheld test case
    - If any part of code crashes: 0 points awarded.
    - testPolicy() returns an in-sample result with cumulative return greater than benchmark: 5 points

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 may 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 and commissions).

Report: 35 points

- Is the method by which the learner is utilized to create a trading strategy described sufficiently clearly that an informed reader could reproduce the result? (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 2000 words? (10 point deduction if so)
- Does the report contain more than 6 figures and/or tables? (10 point deduction if so)
- Is the report especially well written (up to 2 point bonus if so)

Required, Allowed & Prohibited

Required:

- Your project must be coded in Python 2.7.x.
- Your code must run on one of the university-provided computers (e.g. buffet02.cc.gatech.edu).
- All code must be your own.
- 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.
- You may reuse sections of code (up to 5 lines) that you collected from other students or the internet.
- Code provided by the instructor, or allowed by the instructor to be shared.
- Use util.py (only) for reading data.

Prohibited:

- Any libraries not listed in the "allowed" section above.
- Any code you did not write yourself (except for the 5 line rule in the "allowed" section).
- 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

Legacy

- MC3-Project-4-Legacy-Q-trader
- MC3-Project-2-Legacy-trader
- MC3-Project-2-Legacy
