Machine Learning
for Trading

PROJECT 8: STRATEGY EVALUATION

Table of 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.

- May 19, 2022
    - Added the images directory to the provided starter code.

- July 13, 2022
    - Removed README references from sections 3.3.7, 6.2, and 7.1.2

1 OVERVIEW

In this assignment, you implement two strategies and compare their performance. One strategy is a manual strategy, where you will develop the trading rules. The other is a strategy learner, which will develop the trading rules using artificial intelligence. You will submit the code for the project in Gradescope SUBMISSION. You will also submit a report to Canvas.

1.1 Learning Objectives

This project builds on the work of several earlier projects. The specific learning objectives for this assignment are focused on the following areas:

- **Trading Solution**: This project represents the capstone project for the course. This synthesizes the investing and machine learning concepts; and integrates many of the technical components developed in prior projects.
- **Trading Policy Comparison**: Provides an opportunity to evaluate the performance of a manual strategy with that of the AI-learner to better understand its behavior and how it works relative to human-developed strategies.
- **Foundation for Continued Learning**: The knowledge learned and the components throughout the course can serve as a foundation for continued learning and research in trading, machine learning, reinforcement learning, and/or investing.

2 ABOUT THE PROJECT

In this project, you will select a minimum of three and a maximum of all five indicators from Project 6 and use the same indicators in a manual and strategy learner.

2.1 Indicator Selection

- Choose at least 3 indicators. We recommend that these be the same ones researched and reported in P6, however, if you are finding you cannot achieve the results you would like, you may use one new indicator. Note that you can choose only 2 indicators from {SMA, Bollinger Bands, RSI}, just as in P6. Hint: If you're finding poor results using the suggested indicators, be sure to verify your learner's implementations.

- You can only use the indicators that were reported on P6. If you created an EMA function as part of the MACD indicator, you may only use MACD as the indicator and not EMA. You may use EMA if you reported EMA as an indicator.
- Indicators must return a single results vector. As an example, the MACD indicator can only return one vector. This means it must return a custom scalar array that you develop that provides the information you need, the existing Signal line as an array, or the MACD line as an array.
- Indicators can only be used once

2.2 Overall Approach

- Build a **Manual Strategy**, implemented as a class, that combines a minimum of 3 out of the 5 indicators from Project 6.
- Build a **Strategy Learner**, implemented as a class, based on one of the learners described above that uses the same 3+ indicators as used in the manual strategy.
- Test/debug the **Manual Strategy** and **Strategy Learner** on specific symbol/time period problems.
- Conduct experiments.
- Write a report describing your **Manual Strategy**, **Strategy Learner,** and **Experiments**.

2.3 Implement a Strategy Learner

You must draw on the learners you have created so far in the course. Your choices are:

1. **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 convert your regression learner to use mode rather than mean (RTLearner, BagLearner).
2. **Reinforcement-based learner**: 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, which must be used in conjunction with the Classification-based or Reinforcement-based learner. 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., add_evidence()) your learner will be provided with a stock symbol and a time period. It should use this data to learn a strategy. For instance, a classification-based learner 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 the same indicators as you use in the Manual Strategy in Strategy Learner so we can compare your results. You may optimize your indicators for time (vectorization).

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 add_evidence() method. The timeout requirements (see rubric) will be set accordingly. Multiple calls to your testPolicy() method should return exactly the same result.

2.4 Overall Considerations

Overall, your tasks for this project include:

- Build a **Manual Strategy**, implemented as a class, that combines a minimum of 3 out of the 5 indicators from Project 6.
- Build a **Strategy Learner**, implemented as a class, based on one of the learners described above that uses the same 3+ indicators as used in the manual strategy.
- Test/debug the **Manual Strategy** and **Strategy Learner** on specific symbol/time period problems.
- Conduct experiments.

Write a report describing your **Manual Strategy**, **Strategy Learner,** and **Experiments**.

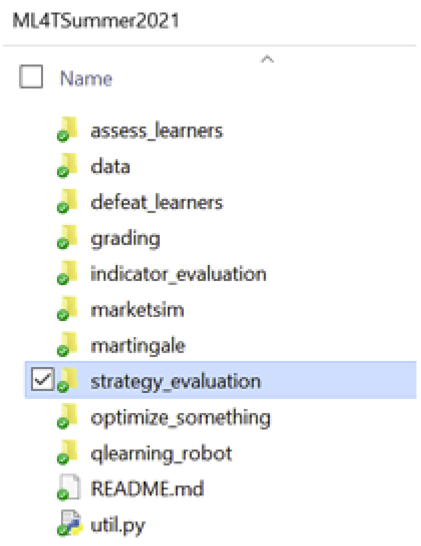
3 YOUR IMPLEMENTATION

Before the deadline, make sure to pre-validate your submission using Gradescope TESTING. Once you are satisfied with the results in testing, submit the code to Gradescope SUBMISSION. **Only code submitted to Gradescope SUBMISSION will be graded. If you submit your code to Gradescope TESTING and have not also submitted your code to Gradescope SUBMISSION, you will receive a zero (0).**

3.1 Getting Started

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 8 can be obtained from: Strategy_Evaluation_2022Summer.zip.

Extract its contents into the base directory (e.g., ML4T_2022Summer). This will add a new folder called "strategy_evaluation" to the course directory structure:



The framework for Project 8 can be obtained in the strategy_evaluation folder alone. Within the strategy_evaluation is the grading script:

- grade_strategy_learner.py
    - Script to test your implementation. This matches the publicly available tests for this project. This file does not need to be turned in and allows you to test locally.

Additional files will also need created and added to the project directory. These files are from previous projects or will need to be created:

- testproject.py
    - Code initializing/running all necessary files for the report. *NOTE: You will have to create the contents of this file yourself.*

- ManualStrategy.py
    - Code implementing a ManualStrategy object (your Manual Strategy) in the strategy_evaluation/ directory. It should implement testPolicy() which returns a trades data frame (see below). The main part of this code should call marketsimcode as necessary to generate the plots used in the report. *NOTE: You will have to create this file yourself.*
- Place your existing Q-Learner, RTLearner, and BagLearner (and DTLearner as well if inheritance is involved), or OptimizationLearner into the strategy_evaluation/ directory.
- Place your existing indicators.py into the strategy_evaluation/ directory
    - *NOTE: You can make changes to the indicators to properly work with both Manual Strategy and Strategy Learner but both strategies must use the same indicator code*. *If you choose to create new indicators, add them to indicators.py.*
- Place your existing marketsimcode.py into the strategy_evaluation/ directory
    - (optional: if needed; note that this is marketsimcode.py and not marketsim.py).
- StrategyLearner.py
    - Code implementing a StrategyLearner object (your ML strategy) in the strategy_evaluation directory. *NOTE: You will have to create this file yourself.*
- experiment1.py and experiment2.py
    - Code conducting the experiments outlined below. *NOTE: You will have to create these files yourself.*

See “what to turn in” below for a list of files that should be submitted.

3.2 Data Details, Dates & 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 compare results more easily. We will test your **Strategy Learner** with other symbols as well.

- You may use data from other symbols (such as SPY) to inform both your **Manual Strategy** and **Strategy Learner**.
- The in-sample 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.
- Only buy/sell actions are allowed. Stops, trailing stops, stop-loss, or any other trading setup is not allowed.
- Benchmark: The performance of a portfolio starting with $100,000 cash, investing in 1000 shares of the symbol in use on the first trading day, and holding that position. Include transaction costs.
- There is no limit on leverage. This means that you do not need to confirm that you have the capital to make your trade. All trades can be executed without validating available cash in your portfolio.
- Transaction costs:
    - ManualStrategy and StrategyLearner: Commission: $9.95, Impact: 0.005 (unless stated otherwise in an experiment).
    - Auto-Grader Commission will always be $0.00, Impact may vary, and will be passed in as a parameter to the learner

3.3 Tasks & Requirementsts

You will implement a manual rule-based trader, the strategy learner, and conduct experiments.

3.3.1 Implement Manual Rule-Based Trader

Not included in the template. You will have to create this code file.

Create ManualStrategy.py and implement a set of rules using at a minimum of 3 indicators you created in Project 6 (NOTE: You can make changes to the indicators to properly work with both **Manual Strategy** and **Strategy Learner** but both strategies must use the same indicator code). Devise some simple logic using your indicators to enter and exit positions in the stock. All indicators must be used in some way to determine a buy/sell signal. You cannot use a single indicator for all signals.

A recommended approach is to create a single logical expression that yields a -1, 0, or 1, corresponding to a “short,” “out” or “long” position. Example usage is signal: If you are out of the stock, then a 1 would signal a BUY 1000 order. If you are long, a -1 would signal a SELL 2000 order. You don't have to follow this advice though, so long as you follow the trading rules outlined above.

For the report we want a written description, not code, however, it is OK to augment your written description with a pseudocode figure.

You should tweak your rules as best you can to get the best performance possible during the in-sample period (do not peek at out-of-sample performance) and should include more than one trade. Use your rule-based strategy to generate a trades DataFrame over the in-sample period.

We expect that your rule-based strategy should outperform the benchmark over the in-sample period.

Benchmark: The performance of a portfolio starting with $100,000 cash, investing in 1000 shares of JPM on the first trading day, and holding that position.

Your ManualStrategy will also need to create two charts.

For the in-sample period:

- **Benchmark** (starting with $100,000 cash, investing in 1000 shares of JPM, and holding that position): **Purple line**
- Performance of **Manual Strategy: Red line**
- Both should be **normalized to 1.0** at the start.
- Vertical **blue lines** indicating LONG entry points.
- Vertical **black lines** indicating SHORT entry points.

For the out-of-sample period:

- Benchmark (starting with $100,000 cash, investing in 1000 shares of JPM, and holding that position): **Purple line**
- Performance of **Manual Strategy: Red line**
- Both should be **normalized to 1.0** at the start.
- Vertical **blue lines** indicating LONG entry points.
- Vertical **black lines** indicating SHORT entry points.

Note: The vertical lines are short and long ***entry*** *points, not buy and sell indicators.*

Create a table that summarizes the performance of the stock, and the Manual Strategy for both in-sample and out-of-sample periods. At a minimum, the table must include:

- Cumulative return of the benchmark and Manual Strategy portfolio
- STDEV of daily returns of the benchmark and Manual Strategy portfolio

- Mean of daily returns of the benchmark and Manual Strategy portfolio

Your ManualStrategy should implement the following API:

```
df_trades = ms.testPolicy(symbol = "AAPL", sd=dt.datetime(2010, 1, 1), ed=dt.date
```

manual_strategy_api.py hosted with ❤ by GitHub view raw

3.3.2 Implement Strategy Learner

For this part of the project, you should develop a learner that can learn a trading policy using your learner and the same indicators used in the Manual Strategy (*NOTE: You can make changes to the indicators to properly work with both Manual Strategy and Strategy Learner but both strategies must use the same indicator code*). You must draw on the learners you have created so far in the course. Your choices are:

1. **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 convert your regression learner to use mode rather than mean in both the RTLearner and BagLearner.
2. **Reinforcement-based learner**: 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., add_evidence()) your learner will be provided with a stock symbol and a time period. It should use this data to learn a strategy. For instance, a classification-based learner 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 the same indicators as you use in the Manual Strategy in Strategy Learner so we can compare your results. You may optimize your indicators for time (vectorization).

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 add_evidence() method. The timeout requirements (see rubric) will be set accordingly. Multiple calls to your testPolicy() method should return exactly the same result.

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 for 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. Remember that impact should be included in the learner's decision process.

Your Strategy Learner should find the optimal parameters **that you choose to optimize** for each indicator and should result in more than one trade. Remember, the indicators used must match those used for Manual Strategy. However, optimization of the indicators does not need to match that of Manual Strategy. Example: for SMA the learner could find the optimal lookback window to use.

NOTE: Lookback windows are not required to be optimized. You can use the same window used in the Manual Strategy if you wish.

Your StrategyLearner should implement the following API:

```
import StrategyLearner as sl
learner = sl.StrategyLearner(verbose = False, impact = 0.0, commission=0.0) # con
learner.add_evidence(symbol = "AAPL", sd=dt.datetime(2008,1,1), ed=dt.datetime(20
df_trades = learner.testPolicy(symbol = "AAPL", sd=dt.datetime(2010,1,1), ed=dt.d
```

strategy_learner_api.py hosted with ❤ by GitHub view raw

The input parameters are:

- verbose: if False do not generate any output
- impact: The market impact of each transaction
- commission: The commission amount charged.
- 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.

3.3.3 Implement Experiment 1

Not included in the template. You will have to create this code file.

Experiment 1 should compare the results of your manual strategy and the strategy learner. It should:

Compare your Manual Strategy with your Strategy Learner in-sample trading JPM. Create a chart that shows:

- Value of the ManualStrategy portfolio (normalized to 1.0 at the start)
- Value of the StrategyLearner portfolio (normalized to 1.0 at the start)
- Value of the Benchmark portfolio (normalized to 1.0 at the start)

Compare your Manual Strategy with your Strategy Learner out-of-sample trading JPM. Create a chart that shows:

- Value of the ManualStrategy portfolio (normalized to 1.0 at the start)
- Value of the StrategyLearner portfolio (normalized to 1.0 at the start)
- Value of the Benchmark portfolio (normalized to 1.0 at the start)

The code that implements this experiment and generates the relevant charts and data should be submitted as experiment1.py.

See DATA DETAILS, DATES & RULES section above for commission and impact information.

3.3.4 Implement Experiment 2

Not included in the template. You will have to create this code file.

Conduct an experiment with your StrategyLearner that shows how changing the value of impact should affect in-sample trading behavior.

Select two metrics, and generate tests that will provide you with at least 3 measurements when trading JPM on the in-sample period with a commission of $0.00. Generate charts that support your tests and show your results.

The code that implements this experiment and generates the relevant charts and data should be submitted as experiment2.py.

See the 'Report' section on Experiment 2 for more details.

3.3.5 Implement Test Project

Execution Limit: 10 minutes

Not included in the template. You will have to create this code file.

Create testproject.py. Testproject.py is the entry point to your project, and it should implement the necessary calls (following each respective API) to Manual Strategy.py, StrategyLearner.py, experiment1.py, and experiment2.py with the appropriate parameters to run everything needed for the report in a single Python call:

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

testproject hosted with ❤ by **GitHub** **view raw**

3.3.6 Implement author() function/method

Deduction if not implemented

You should implement a function called author() that returns your Georgia Tech user ID as a string in all python files. 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.
```

author_example.py hosted with ❤ by **GitHub** **view raw**

3.4 Technical Requirements

The following technical requirements apply to this assignment:

- The file testproject.py must run within 10 minutes.

3.5 Hints and Resources

- You can use util.py to read some other values (e.g., columns other than adjusted close) from the data. Look carefully at util.py and you will see that you can query for other values.
- Your **positions** must be one of three fixed sizes: -1000 shares, +1000 shares, 0 shares.
- You can **trade** up to 1000 or 2000 shares at a time as long as you maintain the requirement of holding 1000, 0, or -1000 shares.
- There is no limit on leverage. This means that you don't need to verify that you have enough cash in your portfolio to make a trade.

4 CONTENTS OF REPORT

Answer the following prompt in a maximum of 10 pages (excluding references) in JDF format. Any content beyond 10 pages will not be considered for a grade. Ten pages is a maximum, not a target; our recommended per-section lengths intentionally add to less than 10 pages to leave you room to decide where to delve into more detail. This length is intentionally set expecting that your submission will include diagrams, drawings, pictures, etc. These should be incorporated into the body of the paper unless specifically required to be included in an appendix.

The JDF format specifies font sizes and margins, which should not be altered. Include charts and tables to support each of your answers. Charts and tables should be generated by the code and saved to files. Charts should be properly annotated with legible and appropriately named labels, titles, and legends. Tables should be properly annotated with column names. When numbers are presented in tables, ensure a sufficient level of numeric precision is provided.

Please address each of these points/questions in your report. You may include additional charts in your submission, but the total number of charts may not exceed 10 charts. The report is to be submitted as **p8_strategyEval_report.pdf**.

At a minimum, the report must contain the following sections:

Introduction: ~ 0.5 pages (Optional)

The report should briefly describe the paper's justification. While the introduction may assume that the reader has some domain knowledge, it should assume that the reader is unfamiliar with the specifics of the assignment. The introduction should also present an initial hypothesis (or hypotheses).

Indicator Overview: ~1 page

Briefly describe the indicators you used to devise your **Manual Strategy** and **Strategy Learner**. You must use a minimum of 3 indicators of the 5 you implemented in Project 6. At a minimum, for each indicator discuss the following:

- Include a brief description of how the indicator is implemented.
- Discuss the parameters for each indicator that are optimized in both Manual Strategy and Strategy Learner.

Hint: If you use Bollinger Bands in Project 6 and want to use that indicator here, you can replace it with BB %B, which should work better with this assignment.

Manual Strategy: ~3 pages

Describe how you combined your indicators to create an overall signal. **Explain** how and why you decide to enter and exit your positions? Why do you believe (or not) that this is an effective strategy?

Compare the performance of your Manual Strategy versus the benchmark for the in-sample and out-of-sample time periods. Provide your charts to support the discussion.

Evaluate the performance of your strategy in the out-of-sample period. Note that you should not train or tweak your approach to this data. You should use the classification learned using the in-sample data only.

Explain WHY these differences occur.

Strategy Learner: ~1.5 pages

The centerpiece of this section 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.
- Describe the hyperparameters, their values, and how they were determined.
- Describe how you discretized (standardized) or otherwise adjusted your data. If this was not performed or necessary, explain why.

Experiment 1 (Manual Strategy / Strategy Learner): ~1.5 pages

Describe your experiment in detail: This includes any assumptions, the initial experimental hypothesis, parameter values, and any other information that would enable an informed reader to set up and repeat the experiment.

Describe, interpret, and summarize the outcome of your experiment, and include your charts from experiment 1. Would you expect this relative result every time with in-sample data? Explain why or why not.

Experiment 2 (Strategy Learner): ~1.5 pages

Provide a hypothesis regarding how changing the value of impact should affect in-sample trading behavior and results (provide at least two metrics, assessing a minimum of 3 different measurements for each metric).

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

References: ~0.25 pages (Optional)

References should be placed at the end of the paper in a dedicated section. Reference lists should be numbered and organized alphabetically by the author's last name. If multiple papers have the same author(s) and year, you may append a letter to the end of the year to allow differentiated in-line text (e.g. Joyner, 2018a and Joyner, 2018b in the section above). If multiple papers have the same author(s), list them in chronological order starting with the older paper. Only works that are cited in-line should be included in the reference list. The reference list does not count against the length requirements.

5 TESTING RECOMMENDATIONS

To test your code, we will invoke each of the functions. You are encouraged to perform any tests necessary to instill confidence that the code will run properly when submitted for grading and will produce the required results. You should confirm that testproject.py runs as expected.

In addition to testing on your local machine, you are encouraged to submit your file to Gradescope TESTING, where some basic pre-validation tests will be performed against the code. **There are two Gradescope TESTING environments; one for the strategy evaluation and another for the testproject.py file**. No credit will be given for coding assignments that do not pass this pre-validation. **Gradescope TESTING does not grade your assignment.** The Gradescope TESTING script is not a complete test suite and does not match the more stringent private grader that is used in Gradescope SUBMISSION. Thus, the maximum Gradescope TESTING score of 60, while instructional, does not represent the minimum score one can expect when the assignment is graded using the private grading script. You are encouraged to develop additional tests to ensure that all project requirements are met.

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

6 SUBMISSION REQUIREMENTS

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.

Assignment due dates in your time zone can be found by looking at the Project in the Assignment menu item in Canvas (ensure your Canvas time zone settings are set up properly). This date is 23:59 AOE converted to your time zone. Late submissions are allowed for a penalty. The times and penalties are as follows:

- -10% Late Penalty: +1 Hour late: submitted by 00:59 AOE (next day)
- -25% Late Penalty: +12 Hours Late: submitted by 11:59 AOE (next day)
- -50% Late Penalty: +24 Hours Late: submitted by 23:59 AOE (next day)
- -100% Late Penalty: > 24+ Late: submitted after 23:59 AOE (next day)

Assignments received after Monday at 23:59 AOE (even if only by a few seconds) are 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.

6.1 Report Submission

Complete your report using the JDF format, then save your submission as a PDF. The report is to be submitted as **p8_strategyEval_report.pdf**. 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(s) to Canvas in PDF format only:

p8_strategyEval _report.pdf

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

You are allowed unlimited submissions of the **p8_strategyEval_report.pdf** file to **Canvas**.

6.2 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. **Only code submitted to Gradescope SUBMISSION will be**

graded. If you submit your code to Gradescope TESTING and have not also submitted your code to Gradescope SUBMISSION, you will receive a zero (0).

Please submit the following file to Gradescope **SUBMISSION**:

Your code as:

- <Implemented Learner>.py
  Note: <Implemented Learner.py> refers to: QLearner.py or RTLearner (okay to submit DTLearner if using inheritance) and BagLearner.py, and OptimizeLearner.py.
- ManualStrategy.py
- StrategyLearner.py
- indicators.py
- experiment1.py
- experiment2.py
- marketsimcode.py (optional if needed)
- testproject.py

Do not submit any other files.

Important: You are allowed a MAXIMUM of five (5) code submissions to Gradescope SUBMISSION.

7 GRADING INFORMATION

Your report is worth 30% of your grade. As such, it will be graded on a 30-point scale coinciding with a rubric design to mirror the questions above. Make sure to answer the questions. The submitted code (which is worth 70% of your grade) is run as a batch job after the project deadline. The code will be graded using a 70-point scale coinciding with a rubric design to mirror the implementation details above. 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 all assignments and exams 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 alone. Regrading will only be undertaken in cases where there has been a genuine error or misunderstanding. Please note that requests will be denied if they are not submitted using the Summer 2022 form or do not fall within the timeframes specified on the Assignment Follow-Up page.

7.1 Grading Rubric

7.1.1 Report [30 points]

The following adjustments will be applied to the report:

Overall Report (max deduction, -30 Points)

- If the report is especially well written (up to +2 point bonus)
- If the strategy does not utilize a minimum of 3 indicators (up to -30 points)
- If the report description does not match the code (up to -10 points)

Indicator Overview (max deduction, -2 Points)

- If the indicators used in Manual Strategy and Strategy Learner are not briefly described (up to -2 points)

Manual Strategy (max deduction, -6 Points)

- If the trading strategy is not described with clarity or with insufficient detail for someone else to reproduce it (up to -4 points)
- If the manual trading system does not provide a higher cumulative return than the benchmark over the in-sample time period (-2 points)
- If differences between the in-sample and out-of-sample performances are not appropriately explained (up to -4 points)
- If the required table is not present or is incorrect (up to -2 points)
- If the student did not use the correct symbol (-2 points)
- If the student did not use the correct date periods (-2 points)
- If the strategy does not obey holding constraints (-6 points)
- Does the provided chart(s) include:
    - Value of benchmark normalized to 1.0 with **purple line** (-1 point, if not)
    - Value of portfolio normalized to 1.0 with **red line** (-1 point, if not)
    - Are vertical lines, appropriately colored, included to indicate entries (-1 point, if not)

Strategy Learner (max deduction, -6 Points)

- If the method by which the learner is utilized to create a trading strategy is not described sufficiently clearly that an informed reader could reproduce the results without referencing your code (up to -5 points)
- If the student chooses to use optimization-based learning and their learning strategy beat the Manual Strategy (+1 bonus point)

Experiment 1 (max deduction, -10 Points)

- If the required experiment is not explained well (up to -7 points)
- If the required experiment is not compellingly supported with the required chart (-3 points)

Experiment 2 (max deduction, -6 Points)

- If the required experiment is not explained well (up to -4 points)
- If the required experiment is not compellingly supported with tabular or graphical data (-2 points)

7.1.2 Code

Code deductions will be applied if any of the following occur:

- If the submitted code indicators.py does not properly reflect the indicators provided in the report (up to -30 points)
- If the indicators used for Manual Strategy and Strategy Learner are not the same as the ones used in Project 6 (up to -30 points)
- If the submitted code and report does not reflect an understanding of the subject matter or does not follow the assignment directions (up to -30 points)
- If the author() method is missing. (up to -5 for each missing instance, with a maximum of -10 points)
- If testproject.py does not produce all included charts in one run and within the time limit without any manipulation (up to -30 points)
- If the charts are not created and saved using Python code.? (up to -30 points) *DO NOT use plt.show() and manually save your charts*.
- If the code saves in a directory outside the project directory. (up to a max of – 30 points

7.1.3 Code & Report

Submission of code and report (up to 100 points deductions):

- If the required code is not provided (including code to recreate the charts and usage of correct trades data frame). (up to -100)
- If the required report is not provided (up to -100)

7.1.4 Auto-Grader (Private Grading Script) [70 points]

We will test Strategy Learner 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
    - add_evidence() 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
    - add_evidence() 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

    - add_evidence() 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
    - add_evidence() 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
- Withheld 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

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

7.1.5 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 add_evidence().

- 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” is **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).

8 DEVELOPMENT GUIDELINES (ALLOWED & PROHIBITED)

See the Course Development Recommendations, Guidelines, and Rules for the complete list of requirements applicable to all course assignments. **The Project Technical Requirements are grouped into three sections: Always Allowed, Prohibited with Some Exceptions, and Always Prohibited**.

The following exemptions to the Course Development Recommendations, Guidelines, and Rules apply to this project:

- N/A