CS7646 ML4T
Machine Learning
for Trading

PROJECT 3: ASSESS LEARNERS

Table of Contents

→ Overview
→ About the Project
→ Your Implementation
→ Contents of Report
→ Testing Recommendations
→ Submission Requirements
→ Grading Information
→ Development Guidelines
→ Optional Resources

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.

- May 25, 2022
    - Added verbiage to 3.5 adding exec() to the forbidden list.

1 OVERVIEW

In this assignment, you will implement four supervised learning machine learning algorithms from an algorithmic family called Classification and Regression Trees (CARTs). You will also conduct several experiments to evaluate the behavior and performance of the learners as you vary one of its hyperparameters. You will submit the code for the project in Gradescope SUBMISSION. You will also submit to Canvas a report where you discuss your experimental findings.

1.1 Learning Objectives

The specific learning objectives for this assignment are focused on the following areas:

- **Supervised Learning**: Demonstrate an understanding of supervised learning, including learner training, querying, and assessing performance.
- **Programming**: Each assignment will build upon one another. The techniques developed here regarding supervised learning and CARTs will play important roles in future projects.
- **Decision Tree Module**: The decision tree(s) implemented in this project will be used in at least one future project.

2 ABOUT THE PROJECT

Implement and evaluate four CART regression algorithms in object-oriented Python: a “classic” Decision Tree learner, a Random Tree learner, a Bootstrap Aggregating learner (i.e, a “bag learner”), and an Insane Learner. As regression learners, the goal for your learner is to return a continuous numerical result (not a discrete result). You will use techniques introduced in the course lectures. However, this project may require readings or additional research to ensure an understanding of supervised learning, linear regression, learner performance, performance metrics, and CARTs (i.e., decision trees).

3 YOUR IMPLEMENTATION

You will implement four CART learners as regression learners: DTLearner, RTLearner, BagLearner, and InsaneLearner. Each of the learners must implement this API specification, where LinRegLearner is replaced by DTLearner, RTLearner, BagLearner, or InsaneLearner, as necessary. In addition, each learner's constructor will need to be revised to align with the instantiation examples provided below.

This project has two main components: First, you will write code for each of the learners and for the experiments required for the report. You must write your own code for this project. You are NOT allowed to use other people's code or packages to implement these learners. Second, you will produce a report that summarizes the observation and analysis of several experiments. The experiments, analysis, and report should leverage the experimental techniques introduced in Project 1.

For the task below, you will mainly be working with the Istanbul data file. This file includes the returns of multiple worldwide indexes for several days in history. In this task, the overall objective is to predict what the return for the MSCI Emerging Markets (EM) index will be based on the other index returns. Y in this case is the last column to the right of the Istanbul.csv file while the X values are the remaining columns to the left (except the first column). As part of reading the data file, your code should handle any data cleansing that is required. This includes the dropping of header rows and date-time columns (i.e., the first column of data in the Istanbul file, which should be ignored). Note that the local *test script* does this automatically for you, but you will have to handle it yourself when developing your implementation.

The Istanbul data is also available here: Istanbul.csv

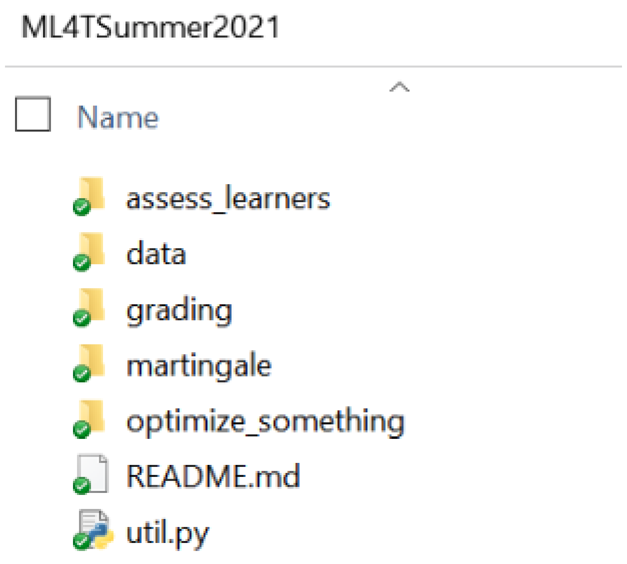
When the grading script tests your code, it randomly selects 60% of the data to train on and uses the other 40% for testing.

The other files, besides Istanbul.csv, are there as alternative sets for you to test your code on. Each data file contains N+1 columns: X1, X2, ... XN, (collectively called the *features*), and Y (referred to as the *target*).

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 3 can be obtained from: Assess_Learners_2022Summer.zip.


ML4TSummer2021

Name

assess_learners
data
grading
martingale
optimize_something
README.md
util.py

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

The framework for Project 3 can be obtained in the assess_learners folder alone. Within the assess_learners folder are several files:

- ./Data (folder)
- LinRegLearner.py
- testlearner.py
- grade_learners.py

The data files that your learners will use for this project are contained in the Data folder. (Note the distinction between the “**d**ata” folder created as part of the local environment and the “**D**ata” folder within the assess_learners folder that will be used in this assignment.)

LinRegLearner is available for your use and must not be modified. However, you can use it as a template for implementing your learner classes. The testlearner.py file contains a simple testing scaffold that you can use to test your learners, which is useful for debugging. It must also be modified to run the experiments. The grade_learners.py file is a local pre-validation script that mirrors the script used in the Gradescope TESTING environment.

You will need to create the learners using the following names: DTLearner.py, RTLearner.py, BagLearner.py, and InsaneLearner.py.

In the assess_learners/Data directory you will find several datasets:

- 3_groups.csv
- ripple_.csv

- simple.csv
- winequality-red.csv
- winequality-white.csv
- winequality.names.txt
- Istanbul.csv

In these files, we have provided test data for you to use in determining the correctness of the output of your learners. Each data file contains N+1 columns: X1, X2, ... XN, and Y.

3.2 Task & Requirements

You will implement the following files:

- DTLearner.py – Contains the code for the regression Decision Tree class.
- RTLearner.py – Contains the code for the regression Random Tree class.
- BagLearner.py – Contains the code for the regression Bag Learner (i.e., a BagLearner containing Random Trees).
- InsaneLearner.py – Contains the code for the regression Insane Learner of Bag Learners.
- testlearner.py – Contains the code necessary to run your experiments and perform additional independent testing.

All your code must be placed into one of the above files. No other code files will be accepted. All files must reside in the assess_learners folder. The testlearner.py file that is used to conduct your experiments is run using the following command:

```
PYTHONPATH=../:. python testlearner.py Data/Istanbul.csv
```

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

3.3 Implement the DT and RT Learners (15 points each)

Implement a Decision Tree learner class named DTLearner in the file DTLearner.py. For this part of the project, your code should build a single tree only (not a forest). You should follow the algorithm outlined in the presentation here: decision tree slides.

- We define “best feature to split on” as the feature (Xi) that has the highest absolute value correlation with Y.

The algorithm outlined in those slides is based on the paper by JR Quinlan which you may also want to review as a reference. Note that Quinlan's paper is focused on creating classification trees, while we are creating regression trees here, so you will need to consider the differences.

You will also implement a Random Tree learner class named RTLearner in the file RTLearner.py. **The RTLearner should be implemented exactly like your DTLearner, except that the choice of feature to split on should be made randomly** (i.e., pick a random feature then split on the median value of that feature). You should be able to accomplish this by revising a few lines from DTLearner (those that compute the correlation) and replacing the line that selects the feature with a call to a random number generator.

The DTLearner and RTLearners will be evaluated against 4 test cases (4 using Istanbul.csv and 1 using another data set from the assess_learners/Data folder). We will assess the absolute correlation between the predicted and actual results for the in-sample data and out-of-sample data with a leaf size of 1, and in-sample data with a leaf size of 50.

3.3.1 Example

The following example illustrates how the DTLearner class methods will be called:

```python
import DTLearner as dt
learner = dt.DTLearner(leaf_size = 1, verbose = False) # constructor
learner.add_evidence(Xtrain, Ytrain) # training step
Ypred = learner.query(Xtest) # query
```

dt_example.py hosted with ❤ by GitHub view raw

The following example illustrates how the RTLearner class methods will be called:

```python
import RTLearner as rt
learner = rt.RTLearner(leaf_size = 1, verbose = False) # constructor
learner.add_evidence(Xtrain, Ytrain) # training step
Ypred = learner.query(Xtest) # query
```

rt_example.py hosted with ❤ by GitHub view raw

The DTLearner and RTLearner constructors take two arguments: leaf_size and verbose. "leaf_size" is a hyperparameter that defines the maximum number of samples to be aggregated at a leaf. If verbose is True, your code can generate output to a screen for debugging purposes. When the tree is constructed recursively, if there are leaf_size or fewer elements at the time of the recursive call, the data should be aggregated into a leaf. Xtrain and Xtest should be NDArrays (Numpy objects) where each row represents an X1, X2, X3… XN set of feature values. The columns are the features and the rows are the individual example instances. Ypred and Ytrain are single dimension NDArrays. Ypred is the prediction based on the given feature dataset. You need to follow the algorithms described above. Do not use a Node Implementation. Your internal representations of the trees must be NDArrays.

3.4 Implement BagLearner (20 points)

Implement Bootstrap Aggregation as a Python class named BagLearner. Your BagLearner class should be implemented in the file BagLearner.py. It should support the API EXACTLY as illustrated in the example below. This API is designed so that the BagLearner can accept any learner (e.g., RTLearner, LinRegLearner, even another BagLearner) as input and use it to generate a learner ensemble. Your BagLearner should support the following function/method prototypes:

```python
import BagLearner as bl
learner = bl.BagLearner(learner = al.ArbitraryLearner, kwargs = {"argument1":1, "
learner.add_evidence(Xtrain, Ytrain)
Y = learner.query(Xtest)
```

baglearner_prototype.py hosted with ❤ by GitHub view raw

The BagLearner constructor takes five arguments: learner, kwargs, bags, boost, and verbose. The learner points to the learning class that will be used in the BagLearner. The BagLearner should support any learner that aligns with the API specification. The "kwargs" are keyword arguments that are passed on to the learner's constructor and they can vary according to the learner (see example below). The "bags" argument is the number of learners you should train using Bootstrap Aggregation. If boost is true, then you should implement boosting (optional implementation). If verbose is True, your code can generate output; otherwise, the code should be silent.

As an example, if we wanted to make a random forest of 20 Decision Trees with leaf_size 1 we might call BagLearner as follows:

```python
import BagLearner as bl
learner = bl.BagLearner(learner = dt.DTLearner, kwargs = {"leaf_size":1}, bags =
learner.add_evidence(Xtrain, Ytrain)
Y = learner.query(Xtest)
```

baglearner_20_dt.py hosted with ❤ by **GitHub** **view raw**

As another example, if we wanted to build a bagged learner composed of 10 LinRegLearners we might call BagLearner as follows:

```
import BagLearner as bl
learner = bl.BagLearner(learner = lrl.LinRegLearner, kwargs = {}, bags = 10, boos
learner.add_evidence(Xtrain, Ytrain)
Y = learner.query(Xtest)
```

baglearner_10_linreg.py hosted with ❤ by **GitHub** **view raw**

Note that each bag should be trained on a different subset of the data. You will be penalized if this is not the case.

Boosting is an optional topic and not required. There is a citation in the Resources section that outlines a method of implementing boosting.

If the training set contains n data items, each bag should contain n items as well. Note that because you should sample with replacement, some of the data items will be repeated.

This code should not generate statistics or charts. If you want to create charts and statistics, you can modify testlearner.py.

You can use code like the below to instantiate several learners with the parameters listed in kwargs:

```
learners = []
kwargs = {"k":10}
for i in range(0,bags):
    learners.append(learner(**kwargs))
```

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

3.5 Implement InsaneLearner (Up to 10-point penalty)

Your BagLearner should be able to accept any learner object so long as the learner obeys the defined API. We will test this in two ways: 1) By calling your BagLearner with an arbitrarily named class and 2) By having you implement InsaneLearner as described below. If your code dies in either case, you will lose 10 points. Note, the grading script only does a rudimentary check thus we will also manually inspect your code for correct implementation and grade accordingly.

Using your BagLearner class and the provided LinRegLearner class; implement InsaneLearner as follows:

- InsaneLearner should contain 20 BagLearner instances where each instance is composed of 20 LinRegLearner instances.

We should be able to call your InsaneLearner using the following API:

```python
import InsaneLearner as it
learner = it.InsaneLearner(verbose = False) # constructor
learner.add_evidence(Xtrain, Ytrain) # training step
Y = learner.query(Xtest) # query
```

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

The InsaneLearner constructor takes one argument: verbose. If verbose is True, your code can generate output; otherwise, the code should be silent. The code for InsaneLearner must be 20 lines or less.

- Each ";" in the code counts as one line.
- Every line that appears in the file (except comments and empty lines) will be counted.
- You can not use the "exec()" statement or strings of Python commands concatenated together using any variant of a new line line '\n'
- Hint: Only include methods necessary to run the assignment tasks and the author methods.

There is no credit for this, but a penalty if it is not implemented correctly. There is a bonus available if your InsaneLearner is implemented in 5 lines or less. Comments, if included, must appear at the end of the file. *Note: We recommend avoiding blank lines and comments in your InsaneLearner implementation.*

3.6 Implement author() (Up to 10-point penalty)

All learners (DT, RT, Bag, Insane) must implement a method called author() that returns your Georgia Tech user ID as a string. This must be explicitly implemented within each individual file and cannot be included through the use of inheritance. It is not your 9 digit student number. Here is an example of how you might implement author() within a learner object:

```
class LinRegLearner(object):
    def __init__(self):
        pass # move along, these aren't the drones you're looking for
    def author(self):
        return 'tb34' # replace tb34 with your Georgia Tech user id.
```

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

And here's an example of how it could be called from a testing program:

```
# create a learner and train it
learner = lrl.LinRegLearner() # create a LinRegLearner
learner.add_evidence(trainX, trainY) # train it
print(learner.author())
```

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

Note: No points are awarded for implementing the author function, but a penalty will be applied if not present.

3.7 Boosting (Optional Learning Activity – 0 points)

This is a personal enrichment activity and will not be awarded any points. Conversely, there is no deduction if boosting is not implemented. Implement boosting as part of BagLearner. How does boosting affect performance compared to not boosting? Does overfitting occur as the number of bags with boosting increases? Create your own dataset for which overfitting occurs as the number of bags with boosting increases.

- Submit your report regarding boosting as report-boosting.pdf

3.8 Implement testlearner.py

Implement testlearner.py to perform the experiment and report analysis as required. This is intended to give a central location to complete the experiments, and produce all necessary outputs, in a single standardized file.

- It is the ONLY file submitted that produces outputs (e.g. charts, stats, calculations).
- It is the ONLY file allowed to read data with the provided data reading routine (i.e., it is not allowed to use util.py for data reading).
- It MUST run in under 10 minutes.

- It MUST execute all experiments, charts, and data used for the report in a single run.

You are able to use a dataset other than Istanbul.csv if not explicitly stated but must adhere to the following run command EXACTLY regardless of the dataset used (you will have to read the other datasets internally):

```
PYTHONPATH=../:. python testlearner.py Data/Istanbul.csv
```

assess_learner_testlearner hosted with ❤ by GitHub view raw

3.9 Technical Requirements

The following technical requirements apply to this assignment

1. You must use a NumPy array (i.e., NDArray()), as shown in the course videos, to represent the decision tree. **You may not use another data structure (e.g., node-based trees) to represent the decision tree.**
2. You may not use Pandas or python list variables within the DTLearner or within the RTLearner.
3. You may not cast a variable of another data type into an NDArray.
4. Although Istanbul.csv is time-series data, **you should ignore the date column and treat the dataset as a non-time series dataset**. In this project, we ignore the time-ordered aspect of the data. In a later project, we will consider time-series data.
5. Files must be read using one of two approaches: 1) The open/readline functions (an example of which is provided in the testlearner.py file) or 2) using NumPy's genfromtxt function (an example of which is used in the grade_learners.py file). Do not use util.py to read the files for this project.
6. The charts must be generated as .png files (when executed using the command given above in these instructions) and should include any desired annotations. Charts must be properly annotated with legible and appropriately named labels, titles, and legends. Image files cannot be post-processed to add annotations or otherwise change the image prior to their inclusion into the report. Charts must be saved to the project directory.
7. The learner code files (i.e., DTLearner, RTLearner, BagLearner, InsaneLearner) should not generate any output to the screen/terminal/display or directly produce any charts. Testlearner.py is the only file that should generate charts.
8. Your learners should be able to handle any number of feature dimensions in X from 2 to N.

9. Performance:
    - DTLearner tests must complete in 10 seconds each (e.g., 50 seconds for 5 tests)
    - RTLearner tests must complete in 3 seconds each (e.g., 15 seconds for 5 tests
    - BagLearner tests must complete in 10 seconds each (e.g., 100 seconds for 10 tests)
    - InsaneLearner must complete in 10 seconds each
10. If the "verbose" argument is True, your code can print out the information for debugging. If verbose = False your code must not generate ANY output other than the required charts. The implementation must not display any text (except for "warning" messages) on the screen/console/terminal when executed in Gradescope SUBMISSION.
11. Any necessary text, tables, or statistics can be saved in a file called p3_results.txt or p3_results.html.
12. Watermarked Charts (i.e., where the GT Username appears over the lines) can be shared in the designated pinned (e.g., "Project 3 – Student Charts") thread alone. Charts presented in reports or submitted for grading must not contain watermark.

3.10 Hints and Resources

"Official" course-based materials:

- How to use a decision tree if you have one (Balch Youtube video)
- How to build a decision tree & Random Trees (Balch Youtube video)
- Media:How-to-learn-a-decision-tree.pdf Balch slides on decision trees
- Decision-tree-example.xlsx Example tabular version of decision tree

Additional supporting materials:

- You may be interested in looking at Andrew Moore's slides on instance-based learning.
- A definition of correlation is used to assess the quality of the learning.
- Bootstrap Aggregating
- AdaBoost
- numpy corrcoef

- numpy argsort
- RMS error

If after submission for grading you are not entirely satisfied with the implementation, you are encouraged to continue to improve the learner(s) as they can play a role in future projects.

4 CONTENTS OF REPORT

In addition to submitting your code to Gradescope, you will also produce a report. The report will be a maximum of 7 pages (excluding references) and be written in JDF Format. Any content beyond 7 pages will not be considered for a grade. At a minimum, the report must contain the following sections:

Abstract

First, include an abstract that briefly introduces your work and gives context behind your investigation. Ideally, the abstract will fit into 50 words, but should not be more than 100 words.

Introduction

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).

Methods

Discuss the setup of the experiment(s) in sufficient detail that **an informed reader (someone with the familiarity of the field, but not the assignment) could set up and repeat the experiment(s)** you performed.

Discussion

The discussion section must include a textual discussion and supporting charts for the following three experiments:

Experiment 1

Research and discuss overfitting as observed in the experiment. (Use the dataset Istanbul.csv with DTLearner). Support your assertion with graphs/charts. (Do not use bagging in Experiment 1). At a minimum, the following question(s) that must be answered in the discussion:

- Does overfitting occur with respect to leaf_size?

- For which values of leaf_size does overfitting occur? Indicate the starting point and the direction of overfitting. Support your answer in the discussion or analysis. Use RMSE as your metric for assessing overfitting.

Experiment 2

Research and discuss the use of bagging and its effect on overfitting. (Again, use the dataset Istanbul.csv with DTLearner.) Provide charts to validate your conclusions. Use RMSE as your metric. At a minimum, the following questions(s) must be answered in the discussion.

- Can bagging reduce overfitting with respect to leaf_size?
- Can bagging eliminate overfitting with respect to leaf_size?

To investigate these questions, choose a fixed number of bags to use and vary leaf_size to evaluate. If there is overfitting, indicate the starting point and the direction of overfitting. Support your answer in the discussion or analysis.

Experiment 3

Quantitatively compare “classic” decision trees (DTLearner) versus random trees (RTLearner). For this part of the report, you must conduct new experiments; do not use the results of Experiment 1. Importantly, RMSE, MSE, correlation, and time to query are not allowed as metrics for this experiment.

Provide at least two new quantitative measures in the comparison.

- Using two similar measures that illustrate the same broader metric does not count as two separate measures. *(Note: Do not use two measures for the accuracy or use the same measurement for two different attributes – e.g., **time** to train and **time** to query are both considered a use of the “**time**” metric.)*
- Provide charts to support your conclusions.

At a minimum, the following question(s) must be answered in the discussion.

- In which ways is one method better than the other?
- Which learner had better performance (based on your selected measures) and why do you think that was the case?
- Is one learner likely to always be superior to another (why or why not)?

Note: Metrics that have been used in prior terms include Mean Absolute Error (MAE), Coefficient of Determination (R-Squared), Mean Absolute Percentage Error (MAPE), Maximum Error (ME), Time, and Space. In addition, please feel free to explore the use of other metrics you discover.

Summary

The summary is an opportunity to synthesize and summarize the experiments. Ideally, it presents key findings and insights discovered during the research. It may also identify interesting areas for future investigation.

5 TESTING RECOMMENDATIONS

To test your code, we will invoke each of the functions. We will also run the testlearner.py file to run your experiments. 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 testlearner.py runs as expected from the assess_learners folder. The following command illustrates how we will run your experiments:

```
PYTHONPATH=../:. python testlearner.py Data/Istanbul.csv
```

assess_learner_testlearner hosted with ❤ by GitHub view raw

Additionally, we provide the grade_learners.py file that can be used for lightweight testing. This local grading/pre-validation script is the same script that will be run when the code is submitted to GradeScope TESTING. To run and test that the file will run from within the assess_learners directory, use the command:

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

pythonpath_grade_learners hosted with ❤ by GitHub view raw

In addition to testing on your local machine, you are encouraged to submit your files to Gradescope TESTING, where some basic pre-validation tests will be performed against the code. **There are two Gradescope TESTING environments; one for the learners and another for the testlearner.py file**. You are encouraged to use both. 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 40, 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(s) should be named **p3_assesslearners_report.pdf** and **report-boosting.pdf** (optional). Assignments should be submitted to the corresponding assignment submission page in Canvas. Please submit the following file(s) to Canvas in PDF format only:

p3_assesslearners_report.pdf

report-boosting.pdf (optional)

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 **p3_assesslearners_report.pdf** and **report-boosting.pdf** (optional) 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 files to Gradescope **SUBMISSION**:

RTLearner.py

DTLearner.py

InsaneLearner.py

BagLearner.py

testlearner.py

Do not submit any other files.

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

7 GRADING INFORMATION

Your report is worth 50% of your grade. As such, it will be graded on a 50-point scale coinciding with a rubric design to mirror the questions above. Make sure to answer those questions. The submitted code (which is worth 50% of your grade) is run as a batch job after the project deadline. The code will be graded using a 50-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 [50 points + up to 3 bonus points]

Deductions will be applied if any of the following occur:

- If the report is not neat or is not well organized (-5 points)
- If the experimental methodology and setup are not well described (up to -5 points per question)
- If the chart does not properly reflect the intent of the assignment (up to -10 points)
- If the chart is not properly labeled, has an incorrect or missing axis, or has an incorrect or missing legend (up to -5 points)
- If all charts provided were not generated in Python (up to -20 points)

- Overfitting / leaf_size question:
  - If one or more charts are not provided to support the argument (up to -5 points)
  - If the student does not state where the region of overfitting occurs (or states that there is no overfitting and that conclusion is not supported by the data) (up to -5 points)
  - If the starting point and direction of overfitting identified is not supported by the data (or if the student states that there is no overfitting and that conclusion is not supported by the data) (up to -5 points)

- Does bagging reduce or eliminate overfitting?:
  - If a chart is not provided to support the argument (up to -5 points)
  - If the student does not state where the region of overfitting occurs (or state that there is no overfitting and that conclusion is not supported by the data) (up to -5 points)
  - If the starting point and direction of overfitting identified is not supported by the data (or if the student states that there is no overfitting and that conclusion is not supported by the data) (up to -5 points)

- Comparison of DT and RT learning
  - If each quantitative experiment is not explained well enough that someone else could reproduce it (up to -5 points)
  - If at least two new quantitative properties are not compared (*reminder – do not use RMSE or correlation*) (-5 points if only one measure is

provided, -10 if no measures are provided)

- If each conclusion regarding each comparison is not supported well with charts (up to -10 points)

- If the student's response indicates a lack of understanding of overfitting (up to -10 points)
- If all charts provided were not generated in Python (up to -20 points)

Bonus Points

- Was the report exceptionally well done? (up to +2 points)
- If InsaneLearner is implemented in 5 lines or less (+1 point)

The report score range is from a minimum of zero (0) to 50 points (+ up to 3 bonus points).

7.1.2 Code

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

- If the author() method is not correctly implemented in each learner file: DTLearner, InsaneLearner, BagLearner, and RTLearner? (-10 points for each missing occurrence)
- If the InsaneLearner is not correctly implemented in 20 lines or less – ***Please remove all blank lines and comments*** (-10 points)
- If the BagLearner does not work correctly with an arbitrarily named class (-10 points)
- If the BagLearner does not generate a different learner in each bag (-10 points)
- If the trees implemented in the DTLearner or RTLearner is not implemented using a NumPy array (-20 points)
- If the DTLearner or RTLearner uses a Python list variable or Pandas (-20 points)
- If the implemented code does not properly reflect the intent of the assignment? (-20 points)
- If the testlearner.py file does not run using the "PYTHONPATH=../:. python testlearner.py Data/Istanbul.csv" statement (-20 points)
- If testlearner.py does not complete running in less than 10 minutes (-20 points)
- If the code does not produce appropriate charts that are saved as .png files. (-20 points)
- If the code displays any charts in a window or screen. (-20 points)

- If the code saves in a directory outside the project directory. (up to a max of –20 points)

7.1.3 Auto-Grader (Private Grading Script) [50 points]

- **DTLearner** in sample/out of sample test, auto-grade 5 test cases (4 using istanbul.csv, 1 using another data set), 3 points each: 15 points.
    - For each test 60% of the data will be selected at random for training and 40% will be selected for testing.
    - Success criteria for each of the 5 tests:
    - Does the correlation between predicted and actual results for **in-sample data** exceed 0.95 with leaf_size = 1?
    - Does the correlation between predicted and actual results for **out-of-sample data** exceed 0.15 with leaf_size=1?
    - Is the correlation between predicted and actual results for **in-sample data** below 0.95 with leaf_size = 50?
    - Does the test complete in less than 10 seconds (i.e. 50 seconds for all 5 tests)?

- **RTLearner** in sample/out of sample test, auto-grade 5 test cases (4 using istanbul.csv, 1 using another data set), 3 points each: 15 points.
    - For each test 60% of the data will be selected at random for training and 40% will be selected for testing.
    - Success criteria for each of the 5 tests:
    - Does the correlation between predicted and actual results for **in-sample data** exceed 0.95 with leaf_size = 1?
    - Does the correlation between predicted and actual results for **out-of-sample data** exceed 0.15 with leaf_size=1?
    - Is the correlation between predicted and actual results for **in-sample data** below 0.95 with leaf_size = 50?
    - Does the test complete in less than 3 seconds (i.e. 15 seconds for all 5 tests)?

- **BagLearner**, auto-grade 10 test cases (8 using istanbul.csv, 2 using another data set), 2 points each 20 points

- For each test 60% of the data will be selected at random for training, 40% will be selected for testing, and leaf_size = 1.
- Success criteria for each run of the 10 tests:
    - For out-of-sample data is a correlation with 1 bag lower than the correlation for 20 bags?
    - Does the test complete in less than 10 seconds (i.e. 100 seconds for all 10 tests)?

The total score is additive, with a minimum score of zero (0) and a maximum score of 50.

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:

- Watermarked charts may be shared in the dedicated discussion forum thread alone.
- You may set a specific random seed for this assignment. If a specific random seed is used, it must only be called once within the testlearner.py file and it must use your GT ID as the numeric value.

9 OPTIONAL RESOURCES

Although the use of these or other resources is not required; some may find them useful in completing the project or in providing an in-depth discussion of the material.

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

James, D. Witten, T. Hastie, R. Tibshirani (2017), An Introduction to Statistical Learning (Chapters 2, 3, and 8)

Murphy, (2021), Probabilistic Machine Learning: An Introduction (Chapters 1, 11, and 18)

Videos:

- Decision Tree Videos, Charles Isbell and Michael Littman, Georgia Tech ML 7641
- Decision Tree Video (Part 1, staring around 0:40 minutes), Tom Mitchell, CMU 601
- Decision Tree Video (Part 2), Tom Mitchell, CMU 601

10 ACKNOWLEDGEMENTS & CITATIONS

The data used in this assignment was provided by UCI's ML Datasets.