Summer 2019 Project 3: Assess Learners

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

You are to implement and evaluate three learning algorithms as Python classes: A "classic" Decision Tree learner, a Random Tree learner, and a Bootstrap Aggregating learner. Note that a Linear Regression learner is provided for you in the git repository. The classes should be named DTLearner, RTLearner, and BagLearner respectively. The Linear Regression learner is already named LinRegLearner.

Note that we are considering this a **regression** problem (not classification). So the goal for your learner is to return a continuous numerical result (not a discrete result). In this project we are ignoring the time order aspect of the data and treating it as if it is static data and time does not matter. In a later project we will make the transition to considering time series data.

You must write your own code for Decision Tree learning, Random Tree learning and bagging. You are NOT allowed to use other people's code to implement these learners.

The project has two main components: The code for your learners, which will be auto graded, and your report, `report.pdf` that should include the components listed below.

Your learners should be able to handle any number of dimensions in X from 2 to N.

New: All of the the decision tree learners you create should be implemented using a matrix data representation (numpy ndarray).

Template and Data

Instructions:

- Download the appropriate zip file File:19spring assess learners.zip
- You should see these files:
  - `assess_learners` the assignment directory
  - `assess_learners/Data/`: Contains data for you to test your learning code on.
  - `assess_learners/LinRegLearner.py`: An implementation of the LinRegLearner class. You can use it as a template for implementing your learner classes.
  - `assess_learners/testlearner.py`: Simple testing scaffold that you can use to test your learners. Useful for debugging.
  - `assess_learners/grade_learners.py`: The grading script; more details here: ML4T_Software_Setup#Running_the_grading_scripts

In the `assess_learners/Data/` directory you will find these files:

- 3_groups.csv
- ripple_.csv
- simple.csv
- winequality-red.csv
- winequality-white.csv
- winequality.names.txt
- istanbul.csv

We will mainly be working with the istanbul data. This data includes the returns of multiple worldwide indexes for a number of days in history. The overall objective is to predict what the return for the MSCI Emerging Markets (EM) index will be on the basis of the other index returns. Y in this case is the last column to the right, and the X values are the remaining columns to the left (except the first column). The first column of data in this file is the date, **which you should ignore.** The grading script does this automatically for you, but you will have to handle it yourself when working on your report.

When the auto grader tests your code we will randomly select 60% of the data to train on and use 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, and Y.

The istanbul data is also available here: File:Istanbul.csv

Implement DTLearner (15 points)

Implement a Decision Tree learner class named DTLearner in the file DTLearner.py. You should follow the algorithm outlined in the presentation here decision tree slides (http://quantsoftware.gatech.edu/images/4/4e/How-to-learn-a-decision-tree.pdf).

- 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 (https://link.springer.com/content/pdf/10.1007/BF00116251.pdf) which you may also want to review as a reference. Note that Quinlan's paper is focused on creating classification trees, while we're creating regression trees here, so you'll need to consider the differences.

For this part of the project, your code should build a single tree only (not a forest). We'll get to forests later in the project. Your code should support exactly the API defined below. DO NOT import any modules besides those listed in the allowed section below. You should implement the following functions/methods:

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

Where "leaf_size" is the maximum number of samples to be aggregated at a leaf. While the tree is being 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. Y and Ytrain are single dimension ndarrays that indicate the value we are attempting to predict with X.

If "verbose" is True, your code can print out information for debugging. If verbose = False your code should not generate ANY output. When we test your code, verbose will be False.

This code should not generate statistics or charts.

Implement RTLearner (15 points)

Implement a Random Tree learner class named RTLearner in the file RTLearner.py. This learner should behave exactly like your DTLearner, except that the choice of feature to split on should be made randomly. You should be able to accomplish this by removing a few lines from DTLearner (the ones that compute the correlation) and replacing the line that selects the feature with a call to a random number generator.

You should implement the following functions/methods:

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

Implement BagLearner (20 points)

Implement Bootstrap Aggregating as a Python class named BagLearner. Your BagLearner class should be implemented in the file `BagLearner.py`. It should support EXACTLY the API defined below. This API is designed so that 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:

```
import BagLearner as bl
learner = bl.BagLearner(learner = al.ArbitraryLearner, kwargs = {"argument1":1, "argument2":2}, bags = 20, boost = False, verb
learner.addEvidence(Xtrain, Ytrain)
Y = learner.query(Xtest)
```

Where learner is the learning class to use with bagging. You should be able to support any learning class that obeys the API defined above for DTLearner and RTLearner. kwargs are keyword arguments to be passed on to the learner's constructor and they 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). 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

```
import BagLearner as bl
learner = bl.BagLearner(learner = dt.DTLearner, kwargs = {"leaf_size":1}, bags = 20, boost = False, verbose = False)
learner.addEvidence(Xtrain, Ytrain)
Y = learner.query(Xtest)
```

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, boost = False, verbose = False)
learner.addEvidence(Xtrain, Ytrain)
Y = learner.query(Xtest)
```

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's 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 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))
```

Implement InsaneLearner (up to 10 point penalty)

Your BagLearner should be able to accept any learner object so long as the learner obeys the API defined above. 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, 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:

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

The code for InsaneLearner should be 20 lines or less. Each ";" in the code counts as one line. All lines (except comments and empty lines) will be counted. There is no credit for this, but a penalty if it is not implemented correctly.

Implement author() Method (up to 10 point penalty)

For all learners you submit (DT, RT, Bag, Insane) should implement a method called `author()` that returns your Georgia Tech user ID as a string. This must be implemented within **each individual file** even if using 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 username.
```

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.addEvidence(trainX, trainY) # train it
    print learner.author()
```

Check the template code for examples. We are adding those to the repo now, but it might not be there if you check right away. Implementing this method correctly does not provide any points, but there will be a penalty for not implementing it.

Experiments and report (50 points)

Create a report that addresses the following questions. Use 11pt font and single spaced lines. We expect that a complete report addressing all the criteria would be at least 3 pages. It should be no longer than 3000 words. To encourage conciseness we will deduct 10 points if the report is too long. The report should be submitted as `report.pdf` in PDF format. Include charts (not tables) to support each of your answers.

- Does overfitting occur with respect to leaf_size? Use the dataset `istanbul.csv` with DTLearner. For which values of leaf_size does overfitting occur? Use RMSE as your metric for assessing overfitting. Support your assertion with graphs/charts. (Don't use bagging).

- Can bagging reduce or eliminate overfitting with respect to leaf_size? Again use the dataset `istanbul.csv` with DTLearner. To investigate this choose a fixed number of bags to use and vary leaf_size to evaluate. Provide charts to validate your conclusions. Use RMSE as your metric.

- Quantitatively compare "classic" decision trees (DTLearner) versus random trees (RTLearner). In which ways is one method better than the other? Provide at least two quantitative measures. Important, using two similar measures that illustrate the same broader metric does not count as two. (For example, do not use two

measures for accuracy.) Note for this part of the report you must conduct new experiments, don't use the results of the experiments above for this.

Note that all charts you provide must be generated in Python (in `testlearner.py`), and you must submit the python code you used to generate the plots.

Hints & resources

"Official" course-based materials:

- How to use a decision tree if you have one (Balch Youtube video) (https://www.youtube.com/watch?v=OBWL4oLT7Uc)
- How to build a decision tree & Random Trees (Balch Youtube video) (https://www.youtube.com/watch?v=WVc3cjvDHhw)
- Media:How-to-learn-a-decision-tree.pdf Balch slides on decision trees
- Media:Decision-tree-example.xlsx Example tabular version of decision tree

Additional supporting materials:

- You may be interested to take a look at Andew Moore's slides on instance based learning (http://www.autonlab.org/tutorials/mbl.html).
- A definition of correlation (http://mathworld.wolfram.com/StatisticalCorrelation.html) which we'll use to assess the quality of the learning.
- Bootstrap Aggregating (https://en.wikipedia.org/wiki/Bootstrap_aggregating)
- AdaBoost (https://en.wikipedia.org/wiki/AdaBoost)
- numpy corrcoef (http://docs.scipy.org/doc/numpy/reference/generated/numpy.corrcoef.html)
- numpy argsort (http://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html)
- RMS error (http://en.wikipedia.org/wiki/Root_mean_square)

Extra Credit (0 points)

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

What to turn in

Be sure to follow these instructions diligently!

Via Canvas, submit as individual files.

- Your code as `RTLearner.py`, `DTLearner.py`, `InsaneLearner.py`, `BagLearner.py`, and `testlearner.py`. Please submit the code (.py) into the assignment "Project 3: Assess Learners (Code)" only.
- Your report as `report.pdf`. Please submit the report (.pdf) into the assignment "Project 3: Assess Learners (Report)" only.

Unlimited resubmissions are allowed up to the deadline for the project.

Rubric

Code (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:
    - 1) Does the correlation between predicted and actual results for **in sample data** exceed 0.95 with leaf_size = 1?
    - 2) Does the correlation between predicted and actual results for **out of sample** data exceed 0.15 with leaf_size=1?
    - 3) Is the correlation between predicted and actual results for **in sample** data below 0.95 with leaf_size = 50?
    - 4) 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:
    - 1) Does the correlation between predicted and actual results for **in sample data** exceed 0.95 with leaf_size = 1?
    - 2) Does the correlation between predicted and actual results for **out of sample** data exceed 0.15 with leaf_size=1?
    - 3) Is the correlation between predicted and actual results for **in sample** data below 0.95 with leaf_size = 50?
    - 4) 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 and 40% will be selected for testing.
  - leaf_size = 20
  - Success criteria for each run of the 10 tests:
    - 1) For out of sample data is correlation with 1 bag lower than correlation for 20 bags?
    - 2) Does the test complete in less than 10 seconds (i.e. 100 seconds for all 10 tests)?
- Is the author() method correctly implemented for all files: DTLearner, InsaneLearner, BagLearner and RTLearner? (-10 points for each if not)
- Is InsaneLearner correctly implemented in 20 lines or less (-10 points if not)
- Does BagLearner work correctly with an arbitrarily named class (-10 points if not)
- Does BagLearner generate a different learner in each bag? (-10 points if not)
- Are the decision tree learners implemented using a matrix? (-20 points if not)
- Does the implemented code properly reflect the intent of the assignment? (-20 points if not)
- Was code used to generate charts submitted? (-20 points if not)

Report (50 points):

- Is the report neat and well organized? (-5 points if not)
- Is the experimental methodology well described (up to -5 points if not)
- Does the chart properly reflect the intent of the assignment? (up to -10 points if not)
- Does the chart include properly labeled axis and legend? (up to -5 points if not)

- Overfitting / leaf_size question:
  - Is one or more charts provided to support the argument? (up to -5 points if not)
  - Does the student state where the region of overfitting occurs (or state that there is no overfitting)? (-5 points if not)

    - Are the starting point and direction of overfitting identified supported by the data (or if the student states that there is no overfitting, is that supported by the data)? (-5 points if not)

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

- Comparison of DT and RT learning
    - Is each quantitative experiment explained well enough that someone else could reproduce it (up to -5 points if not)
    - Are there at least two new quantitative properties that are compared (do not use correlation)? (-5 points if only one, -10 if none)
    - Is each conclusion regarding each comparison supported well with charts? (up to -10 points if not)

- Was the report exceptionally well done? (up to +2 points)
- Does the student's response indicate a lack of understanding of overfitting? (up to -10 points)
- Were all charts provided generated in Python? (up to -20 points if not)

Required, Allowed & Prohibited

Required:

- Your code must implement a Random Tree learner.
- Your project must be coded in Python 2.7.x.
- Your code must run on one of the university-provided computers (e.g. buffet01.cc.gatech.edu).
- Your code must run in less than 10 seconds on one of the university-provided computers.
- The code you submit should NOT include any data reading routines. The provided testlearner.py code reads data for you.
- The code you submit should NOT generate any output: No prints, no charts, etc. *Note that all charts you provide must be generated in Python (in testlearner.py)

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.
- Cheese.

Prohibited:

- Any other method of reading data besides testlearner.py
- 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).
- Code that includes any data reading routines. The provided testlearner.py code reads data for you.
- Code that generates any output when verbose = False: No prints, no charts, etc.

FAQ

- Q: Can I use an ML library or do I have to write the code myself? A: You must write the decision tree and bagging code yourself. The LinRegLearner is provided to you. Do not use other libraries or your code will fail the auto grading test cases.

- Q: Which libraries am I allowed to use? Which library calls are prohibited? A: The use of classes that create and maintain their own data structures are prohibited. So for instance, use of `scipy.spatial.KDTree` is not allowed because it builds a tree and keeps that data structure around for reference later. The intent for this project is that YOU should be building and maintaining the data structures necessary. You can, however, use methods that return immediate results and do not retain data structures
    - Examples of things that are allowed: sqrt(), sort(), argsort() -- note that these methods return an immediate value and do not retain data structures for later use.
    - Examples of things that are prohibited: any scikit add on library, scipy.spatial.KDTree, importing things from libraries other than pandas, numpy or scipy.

- Q: How should I read in the data? A: Your code does not need to read in data, that is handled for you in the testlearner.py and grade_learners.py code. For testing your code you can modify testlearner.py to read in different datasets, but your solution should NOT depend on any special code in testlearner.py

- Q: How many data items should be in each bag? A: If the training set is of size N, each bag should contain N items. Note that since sampling is with replacement some of the data items will be repeated.

Acknowledgements and Citations

The data used in this assignment was provided by UCI's ML Datasets (http://archive.ics.uci.edu/ml/datasets/ISTANBUL+STOCK+EXCHANGE).

Legacy

- MC3-Project-1-legacy
- MC3-Project-1-legacy
