MC3-Project-1

From Quantitative Analysis Software Courses

Contents

Updates / FAQs

- **2017-02-10**
  - Information regarding data source is updated in "Template and Data" section.
  - Details regarding author() method are fleshed out a bit.

- **2017-02-09**
  - switch to new data source, istanbul.csv
  - author() method requirement added
  - rubric and experiment requirements updated

- 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 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.

Overview

You are to implement and evaluate three learning algorithms as Python classes: A Random Tree learner, a Linear Regression learner (provided for you) and a Bootstrap Aggregating learner. The classes should be named RTLearner, LinRegLearner, and BagLearner respectively. You can use the provided testlearner.py code as a framework for testing your code, we will use similar code in our autograder to test your learners. Be sure that your solution does not depend on any code in testlearner.py

We are considering this a **regression** problem (not classification). So the goal is to return a continuous numerical result (not a discrete result). In this project we are ignoring the time 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 time series data.

You must write your own code for Random Tree learning and bagging. You are NOT allowed to use other peoples' code to implement Random Trees or bagging.

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 learner should be able to handle any dimension in X from 2 to N.

Reference Material

"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)
- paper on Random Trees by Adele Cutler (http://www.interfacesymposia.org/I01/I2001Proceedings/ACutler/ACutler.pdf)
- 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)

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))
```

Template and Data

Instructions:

- Update your copy of the class' github repo. We will send separate instructions by email on how to do that.

You will find these files in the mc3_p1 directory

- `Data/`: Contains data for you to test your learning code on.
- `LinRegLearner.py`: An implementation of the LinRegLearner class. You can use it as a template for implementing your learner classes.
- `testlearner.py`: Helper code to test a learner class.

In the 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.**

When we test your code we will randomly select 60% of the data to train on and use the other 40% for testing. However, as of this writing, the testlearner.py code uses the **first 60% of the data for training**, and the **remaining 40% for testing**. That may be helpful because it will enable you to compare results with your friends on piazza.

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

Part 1: Implement RTLearner (40%)

You should implement a Random Tree learner class in the file `RTLearner.py`. You should consult the paper by Adele Cutler (http://www.interfacesymposia.org/I01/I2001Proceedings/ACutler/ACutler.pdf) as a reference. Note that for this part of the project, your code should only build a single tree (not a forest). We'll get to forests later in the project. The primary differences between Cutler's Random Tree and the methodology originally proposed by JR Quinlan (https://wwwold.cs.umd.edu/class/fall2009/cmsc828r/PAPERS/fulltext_Quilan_Ashwin_Kumar.pdf) are:

1. The feature i to split on at each level is determined randomly. It is not determined using information gain or correlation, etc.
2. The split value for each node is determined by: Randomly selecting two samples of data and taking the mean of their Xi values.

Your code should support exactly the API defined below. DO NOT import any modules besides those listed in the prohibited/allowed section below. 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
```

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. You may modify testlearner.py to generate statistics and charts.

Part 2: Implement BagLearner (20%)

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 = rt.RTLearner, kwargs = {"leaf_size":1}, bags = 20, boost = False, verbose = False)
learner.addEvidence(Xtrain, Ytrain)
Y = learner.query(Xtest)
```

Where learner is the learning class to use with bagging. kwargs are keyword arguments to be passed on to the learner's constructor and they vary according to the learner (see hints above). "bags" is the number of learners you should train using Bootstrap Aggregation. If boost is true, then you should implement boosting.

If verbose is True, your code can generate output. Otherwise it should be silent.

Notes: See hints section above for example code you might use to instantiate your learners. Boosting is an optional topic and not required. There's a citation below in the Resources section that outlines a method of implementing bagging. 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, modify testlearner.py for that purpose.

Part 3: Implement InsaneLearner (up to 10% penalty)

Using your BagLearner class and the provided LinRegLearner class, implement InsaneLearner as follows: InsaneLearner should contain 20 bagged learners where each of these learners is composed of 20 bagged LinRegLearners. 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 less than 20 lines. There is no credit for this, but a penalty if it is not implemented correctly.

Part 4: Implement author() Method (up to 10% penalty)

For BOTH BagLearner.py and RTLearner.py you should implement a method called `author()` that returns your Georgia Tech user ID as a string. This is the ID you use to log into t-square. 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.

Part 5: Experiments and report (50%)

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 6 pages including charts, tables and text. To encourage conciseness we will deduct 2% for each page over 6 pages. The report should be submitted as `report.pdf` in PDF format. Do not submit word docs or latex files. Include data as tables or charts to support each of your answers.

- Does overfitting occur with respect to leaf_size? Consider the dataset `istanbul.csv` with RTLearner. 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? Fix the number of bags and vary leaf_size to investigate. Provide charts and or tables to validate your conclusion.

- Does overfitting occur with respect to number of bags? Choose some leaf_size and keep it fixed. How does RMSE vary as you increase the number of bags? Does overfitting occur with respect to the number of bags? Support your assertion with graphs/charts.

Hints & resources

Some external resources that might be useful for this project:

- 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)

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))
```

What to turn in

Be sure to follow these instructions diligently!

Via T-Square, submit as attachment (no zip files; refer to schedule for deadline).

- Your code as `RTLearner.py` `InsaneLearner.py` and `BagLearner.py`.
- Your report as `report.pdf`

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

Extra Credit (0%)

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

Rubric

Code (60 points):

- RTLearner in sample/out of sample test, auto grade 10 test cases (8 using istanbul.csv, 2 using another data set), 4 points each: 40 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 10 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. 30 seconds for all 10 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 5 seconds (i.e. 50 seconds for all 10 tests)?
- Is the author() method correctly implemented for BagLearner and RTLearner? (-10% for each if not)
- Is InsaneLearner correctly implemented and 20 lines or less (-10% if not)

Report (40 points):

- Is the report neat and well organized? (-5 points if not)
- Is the experimental methodology well described (-5 points if not)

- Overfitting / leaf_size question:
    - Is data (either a chart or table) 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)

- Overfitting / leaf_size fixed, change number of bags:
    - Is data (either a chart or table) 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)

- Overfitting / bagging fixed, change leaf_size:
    - Is data (either a chart or table) 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)

- Was the report exceptionally well done? (+1%)

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. buffet02.cc.gatech.edu).
- Your code must run in less than 5 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.

Allowed:

- You can develop your code on your personal machine, but it must also run successfully on one of the university provided machines or virtual images.
- Your code may use standard Python libraries.
- You may use the NumPy, SciPy, matplotlib and Pandas libraries. Be sure you are using the correct versions.
- You may reuse sections of code (up to 5 lines) that you collected from other students or the internet.
- Code provided by the instructor, or allowed by the instructor to be shared.
- 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 (except for the 5 line rule in the "allowed" section).
- Any Classes (other than Random) that create their own instance variables for later use (e.g., learners like kdtree).
- 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.

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