Summer 2016 Project 2

From Quantitative Analysis Software Courses

Contents

FAQs from Previous Semesters

- Q: Can I use an ML library or do I have to write the code myself? A: You must write the KNN 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.

- Your strategy for defeating KNNLearner and LinRegLearner should not depend on they way you select training data versus testing data. The relationship of one learner performing better than another should persist regardless

of which 60% of the data is selected for training and which 40% is selected for testing.

- 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. You can modify testlearner.py to read in different datasets. 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 KNN learner, a Linear Regression learner (provided) and a Bootstrap Aggregating learner. The classes should be named KNNLearner, LinRegLearner, and BagLearner respectively. 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 training & testing with static spatial data. In the next project we will make the transition to time series data.

You must write your own code for KNN and bagging. You are NOT allowed to use other peoples' code to implement KNN 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 1 to N.

Template and Data

Instructions:

- Download `mc3_p1.zip`, unzip inside `ml4t/`

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.
- `__init__.py`: Tells Python that you can import classes while in this directory.
- `testlearner.py`: Helper code to test a learner class.

In the Data/ directory there are three files:

- 3_groups.csv
- ripple_.csv
- simple.csv

We will mainly be working with ripple and 3_groups. Each data file contains 3 columns: X1, X2, and Y. In most cases you should use the first 60% of the data for training, and the remaining 40% for testing.

Part 1: Implement KNNLearner (30%)

Your KNNLearner class should be implemented in the file KNNLearner.py. It should implement EXACTLY the API defined below. DO NOT import any modules besides allowed below. You should implement the following functions/methods:

```
import KNNLearner as knn
learner = knn.KNNLearner(k = 3, verbose = False) # constructor
learner.addEvidence(Xtrain, Ytrain) # training step
Y = learner.query(Xtest) # query
```

Where "k" is the number of nearest neighbors to find. 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. Otherwise your code should not generate ANY output.

Use Euclidean distance. Take the mean of the closest k points' Y values to make your prediction. If there are multiple equidistant points on the boundary of being selected or not selected, you may use whatever method you like to choose among them.

This code should not generate statistics or charts. 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 implement EXACTLY the API defined below. You should implement the following functions/methods:

```
import BagLearner as bl
learner = bl.BagLearner(learner = knn.KNNLearner, kwargs = {"k":3}, bags = 20, boo
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 below). "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 below 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. Modify testlearner.py to generate statistics and charts.

Part 3: Experiments and report (50%)

Create a report that addresses the following issues/questions. Use 11pt font and single spaced lines. We expect that a complete report addressing all the criteria would be at least 4 pages. It should be no longer than 10 pages including charts, tables and text. To encourage conciseness we will deduct 2% for each page over 10 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 your answers. I expect that this report will be 4 to 10 pages.

- Include charts or tables of data to support your results. However your submitted code should not generate statistics or charts. Modify testlearner.py to generate statistics and charts.
- Create your own dataset generating code (call it `best4linreg.py`) that creates data that performs significantly better with LinRegLearner than KNNLearner. Explain your data generating algorithm, and explain why LinRegLearner performs better. Your data should include at least 2 dimensions in X, and at least 1000 points. (Don't use bagging for this section).
- Create your own dataset generating code (call it `best4KNN.py`) that creates data that performs significantly better with KNNLearner than LinRegLearner. Explain your data generating algorithm, and explain why KNNLearner performs better. Your data should include at least 2 dimensions in X, and at least 1000 points. (Don't use bagging for this section).
- Consider the dataset `ripple` with KNN. For which values of K does overfitting occur? (Don't use bagging).

- Now use bagging in conjunction with KNN with the ripple dataset. Choose some K keep it fixed. How does performance vary as you increase the number of bags? Does overfitting occur with respect to the number of bags?
- Can bagging reduce or eliminate overfitting with respect to K for the ripple dataset? Fix the number of bags and vary K.

Python Hint

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 KNNLearner.py, BagLearner.py, best4linreg.py, best4KNN.py
- Your report as report.pdf

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

Extra Credit (3%)

Implement boosting as part of BagLearner. How does boosting affect performance for ripple and 3_groups data?

Does overfitting occur for either of these datasets as the number of bags with boosting increases?

Create your own dataset for which overfitting occurs as the number of bags with boosting increases.

Describe and assess your boosting code in a separate report.pdf. Your report should focus only on boosting. It should be submitted separately to the "extra credit" assignment on t-square.

Rubric

- KNNLearner, auto grade 10 test cases (including ripple.csv and 3_groups.csv), 3 points each: 30 points

- BagLearner, auto grade 10 test cases (including ripple.csv and 3_groups.csv), 2 points each: 20 points
- best4linreg.py
  - Code submitted: -5 if absent
  - Description complete -- Sufficient that someone else could implement it: -5 if not
  - Description compelling -- The reasoning that linreg should do better is understandable and makes sense. Graph of the data helps but is not required if the description is otherwise compelling: -5 if not
  - Train and test data drawn from same distribution: -5 if not
  - Performance demonstrates that linreg does better: -10 if not
- best4KNN.py
  - Code submitted: -5 if absent
  - Description complete -- Sufficient that someone else could implement it: -5 if not
  - Description compelling -- The reasoning that KNN should do better is understandable and makes sense. Graph of the data helps but is not required if the description is otherwise compelling: -5 if not
  - Train and test data drawn from same distribution: -5 if not
  - Performance demonstrates that KNN does better: -10 if not
- Overfitting
  - Student conveys a correct understanding of overfitting in the report?: -5 points if not.
  - Is the region of overfitting correctly identified? -5 points if not.
  - Is the conclusion supported with data (table or chart): -5 points if not.
- Bagging
  - Correct conclusion regarding overfitting as bags increase, supported with tables or charts: -10 points if not.
  - Correct conclusion regarding overfitting as K increases, supported with tables or charts: -10 points if not.

Test Cases

The following test cases may be of use. They are from a previous semester's grading test cases.

- MC3-Project-1-Test-Cases-spr2016

Required, Allowed & Prohibited

Required:

- Your project must be coded in Python 2.7.x.

- Your code must run on one of the university-provided computers (e.g. buffet02.cc.gatech.edu), or on one of the provided virtual images.
- 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: No prints, no charts, etc.
