Problem Set 8: Space Cows

Handed out: Thursday, April 18, 2013
Due: 11:59pm, Thursday, April 25, 2013

Introduction

A colony of Aucks -- superintelligent alien bioengineers -- has landed on Earth and decided to create a new species of cow that can jump over the moon! The Aucks are performing their experiments on Earth, and if successful, plan on transporting the mutant cows back to their home planet of Aurock. In this problem set, you will use regression analysis to model the aliens' cow breeding patterns, and implement algorithms to figure out how the aliens should shuttle their experimental cows back across outer space.

Getting Started

Download pset8.zip, which contains the data and skeleton code you will use in this problem set.

Part A: Breeding Alien Cows

You don't know what strange experimental breeding techniques the Aucks are using, but with the help of your friends Eric and Joanna, you have observed and collected data on how many cows are in the alien herd at the end of each day.
`ps8a_data1.txt` and `ps8a_data2.txt` contain 50 days worth of data that you and Joanna collected on alternating days. `ps8a_data3.txt` contains 100 days worth of data collected by Eric, who lives across town but observed a similar alien invasion near his home.
Backed by this information, you now feel confident that you can find an equation to model the alien cow breeding pattern.

Problem 1: File I/O

Fill in the function `loadData(filename)`that takes the name of a text file as a string and reads in its contents. You can expect the data to be formatted as pairs of comma-separated numbers `x,y`, where the first number on each line indicates the number of days since you started observing the aliens, and the second number indicates the number of cows the herd consists of at the end of that day. Here are the first few lines of `ps8a_data3.txt`, for reference:

```
1,89 2,648 3,153 4,269 5,276 ...
```

Because you will be plotting this data later, save all of the x values into one Pylab array and all the y values into a second Pylab array. Your function should return a tuple containing the two arrays.
Hint: Both code from recitation 8 and helper functions in problem sets 3 contain code to read and process data from files. This is similar to code that you will write for this problem, if you feel like you need further reference.
In the given code skeleton, there is a `main` function. After completing the `loadData` function, complete the three lines in the main function to load data from`ps8a_data1.txt`, `ps8a_data2.txt`, and `ps8a_data3.txt`. You should be calling the `loadData` function you just wrote. Once you have done this step correctly, run`ps8a.py` to make sure it passes the first two assertions in the `main` function.

Problem 2: Curve Fitting

Now that we have data, we can start fitting curves to our data points using the Pylab module. In this problem, we will first write two helper functions for two important types of curves: polynomial and exponential. With these functions, we will be able to generate many different models to evaluate which is the best fit.

Problem 2a

Implement the function `polyFit(x, y, degree)`in `ps8a.py`. This function should take as input two Pylab arrays holding x and y values, as well as the integer degree of a polynomial. Calculate the best fit polynomial curve of the input degree on the given data, and return the coefficients of the polynomial as an array (the order of the coefficients is up to you). For full points, your method should be as concise as possible and use built in Pylab functionality.

Problem 2b

Implement the function `expFit(x, y)`. Again, this function takes as input two Pylab arrays x and y. Calculate the best fit base 2 exponential curve for x and y. **NOTE: the base of the exponential does not matter for this problem, as we are simply transforming the data to linearize it, and transforming it back. If you feel comfortable with e and the natural log, feel free to use the corresponding Pylab functions.** This can be done easily if we consider that exponential curves take the following form: $y = a * 2^{b*x}$. This exponential equation can be made linear by taking the log of both sides. This gives `log(y) = log(a) + b*x`, which looks like the familiar linear equation `y = b*x + c` using log(y) instead of y and log(a) instead of c! Therefore, an exponential fit curve can be found by generating the linear regression of log(y) with respect to x.
Your implementation of `expFit` should return the coefficients `b, log_a` of this linear function as an array. In order to undo this transformation for plotting later, note that you can retrieve `a` with $a = 2^{log\_a}$, and solve $y = a * 2^{b*x}$.
You will find the `pylab.log2()` and `pylab.exp2()` functions useful. For full points, reuse your function from Problem 2a.

Problem 3: Evaluating regression functions

Recall that it is common practice when developing scientific models to split data into two or more sets, where the first set is used as *training* data to develop the model, and the remaining sets are used as *testing* data to evaluate the model. By using separate training and testing data sets, we can increase confidence in the correctness of our models.
We will designate `ps8a_data1.txt` as our training data set, generate different fit curves based on this data, and evaluate each curve based on how good of a fit it is by testing it on the other two data sets. Recall that the data from these three data sets have been saved as `(xdata, ydata)` tuples in the `main` function as`data1`, `data2`, and `data3`. In this section, your code should be called from within the `main` function, but you will find it both helpful and useful to declare additional helper functions. *You will be graded both for correctness and code organization*.
Generate the following four fit curves for `data1` by calling the functions you wrote in Problem 2.

1. Linear
2. Quadratic
3. Quartic (polynomial of degree 4)
4. Exponential

These four curves are the four *models* that you will test on `data2` and `data3`. Using these models, calculate the coefficient of determination (R^2) for each model on`data1` using the `rSquare` function provided.
Now, *without regenerating the models*, design plots that display the points stored in `data1`, `data2`, and `data3` as scatter plots against the curve represented by each of the models. (Note: Please plot the data on a linear scale. Do not use log scale!) Also calculate the coefficient of determination for each data set against each model. Verify visually that the curves being plotted make sense (i.e. do they actually fit the data and are they the correct shape?), and that the value of R^2 is correlated to the goodness of the fit that you observe for each graph.

In a separate writeup namedps8.pdf, paste images of your plots and include the coefficient of determination for each. However you decide to present them, you should have 4 fit curves x 3 data sets = 12 plots overall.

Problem 4: Choosing a Model

Given the information you have discovered, which equation would you choose to model the Aucks' mutant cow breeding pattern and why? Feel free to play around with other values for the polynomial degree. Using the model you chose, about how many cows will the aliens have after 200 days? Does this make sense? If it doesn't, choose a different model that does and explain why it's better. Write your answers to these questions as a short paragraph in `ps8.pdf`.

Part B: Transporting Cows Across Space

The aliens have succeeded in breeding cows that jump over the moon! Now they want to take home the top ten cows. These cows all weigh different amounts, and a single alien spaceship pod can only carry **1.0 ton of cargo**. The aliens want to take all chosen cows back, but they want to minimize the number of trips they have to take across the universe. You can assume that all the cows are between 0.0 and 1.0 ton in weight.
The data for these 10 cows is stored in `ps8b_data.txt`. **All of your code from Part B should go into `ps8b.py`**

Problem 5: Loading the Data

Write the `loadCows` function. It should be very similar to the `loadData` function from Part A.

```
def loadCows(filename):

      """

      Read the contents of the given file.  Assumes the file contents contain

      data in the form of comma-separated cow name, weight pairs, and return a

      dictionary containing cow names as keys and corresponding weights as
values



      Parameters:


      filename - the name of the data file as a string



      Returns:

            a dictionary of cow name, weight pairs

      """
```

Problem 6: Greedy Loading

Implement a greedy algorthm for transporting the cows back across space in `greedyTransport`. The result should be a list of lists, where the inner lists contain the names of cows for each trip.

```
def greedyTransport(cows,limit):

      """

      Finds the allocation of cows that minimizes the number of spaceship
trips

      via a greedy heuristic (always choose the heaviest cow to fill the

      remaining space).

      Parameters:

      cows - a dictionary of name (string), weight (float) pairs

      limit - weight limit of the spaceship

      Returns:

      A list of lists, with each inner list containing the names of cows

      transported on a particular trip and the overall list containing all the


      trips

      """
```

Problem 7: Brute Force Loading

Implement a brute force algorithm to find the minimum number of trips needed to take all the cows across the universe. The result should be a list of lists, where the inner lists contain the names of cows for each trip.

```
def bruteForceTransport(cows,limit):

      """

      Finds the allocation of cows that minimizes the number of spaceship
trips

      via brute force



     Parameters:

       cows - a dictionary of name (string), weight (float) pairs

      limit - weight limit of the spaceship
      Returns:

      A list of lists, with each inner list containing the names of cows

      transported on a particular trip and the overall list containing all the

      trips

      """
```

Hint: to generate all the possibilities for the brute force method, you will want to work with set partitions (http://en.wikipedia.org/wiki/Set_partition). For instance, all the possible 2-partitions of the list [1,2,3,4] are [[1,2],[3,4]], [[1,3],[2,4]], [[2,3],[1,4]], [[1],[2,3,4]], [[2],[1,3,4]], [[3],[1,2,4]], [[4],[1,2,3]].
To help you with creating partitions, we have included a helper function `getPartitions(list)` that takes as input a list and returns a *generator* that contains all the possible partitions of this list, from 0-partitions to n-partitions, where n is the length of this list.
Generators are outside the scope of this course, but if you're curious, you can read more about it here: http://wiki.python.org/moin/Generators.
To use generators, you must iterate over the generator to retrieve the elements; **you cannot index into a generator**! For instance, the recommended way to call `getPartitions` on a list [1,2,3] is the following:

```
for part in getPartitions([1,2,3]):



    print part
```

Try out this snippet of code to see what is printed!
Though we're working with lists, you can assume that order doesn't matter -- that is, [[1,2],[3,4]] and [[2,1],[3,4]] are considered the same partitions of [1,2,3,4].

Problem 8: Comparing the Loading Algorithms

Run your greedy and brute force algorithms `ps8b_data.txt` to find the minimum number of trips under each method. How do the results compare? Which algorithm ran faster? You can measure the time a block of code takes to execute using the `time.time()` function as follows:

```
    start = time.time()

      ## a block of code

      end = time.time()

      print end - start
```

This will print the duration in seconds.
Write a brief paragraph with your observations as part of `ps8.pdf`.

```
if __name__ == "__main__":

      """

      Using the data from ps8b_data.txt and the specified weight limit, run
your
```

```
    greedyTransport and bruteForceTransport functions here. Print out the

    number of trips returned by each method, and how long each method takes

    to run in seconds.

    """
```

The end!

http://abstrusegoose.com/406

Hand-In Procedure

1. Save

Save your solutions as **`ps8a.py`** and **`ps8b.py`**. Submit your graphs from Part A and answers from Part B as **`ps8.pdf`**

2. Time and Collaboration Info

At the start of each file, in a comment, write down the number of hours (roughly) you spent on the problems in that part, and the names of the people you collaborated with. For example:

```
# Problem Set 8

  #Name: Jane Lee

  # Collaborators: John Doe

  # Time: 3:30

  #

  ... your code goes here ...
```

3. Sanity checks

After you are done with the problem set, do sanity checks. Run the code and make sure it can be run without errors.

4. Submit

Upload all your files to Stellar. If there is some error uploading to your workspace, email the file to `6.00-staff [at] mit.edu`.

You may upload new versions of each file until the 11:59pm deadline, but anything uploaded after that will be ignored, unless you still have enough late days left.