CSE 4309 - Assignments - Assignment 3

List of assignment due dates.

The assignment should be submitted via Canvas. Submit a file called assignment3.zip, containing at least the following two files:

- answers.pdf, for your answers to the written tasks, and for the output that the programming task asks you to include. Only PDF files will be accepted. All text should be typed, and if any figures are present they should be computer-generated. Scans of handwriten answers will NOT be accepted.
- linear_regression.py, containing your Python code for the programming part. Your Python code should run on the versions specified in the syllabus, unless permission is obtained via e-mail from the instructor or the teaching assistant. Also submit any other files that are needed in order to document or run your code (for example, additional source code files, but NOT data files used as input).

These naming conventions are mandatory, non-adherence to these specifications can incur a penalty of up to 20 points.

Your name and UTA ID number should appear on the top line of both documents.

Task 1 (60 points, programming)

In this task you will implement linear regression, as described in Section 3.1.4 of the PRLM textbook. Note that Section 3.1.4. describes the "batch learning" regularized least-squares approach, and that is the approach you need to implement.

Function Name and Arguments

File linear_regression_main.py contains incomplete code that aims to learn a linear regression model given some training data, and then evaluate this model on test data.

To complete that code, you must create a file called `linear_regression.py`, where you implement a Python function called `linear_regression`. Your function should use linear regression (following the description in the textbook and slides) to fit a polynomial function to the data. In particular, your function should be invoked as follows:

```
linear_regression(training_file, test_file, degree, lambda)
```

- The first argument, `training_file`, is the path name of the training file, where the training data is stored. The path name can specify any file stored on the local computer.
- The second argument, `test_file`, is the path name of the test file, where the test data is stored. The path name can specify any file stored on the local computer.

- The third argument, `degree` is an integer between 1 and 10. We will not test your code with any other values. The degree specifies what function φ you should use. Suppose that you have an input vector $x = (x_1, x_2, ..., x_D)^{T,}$.
  - If the degree is 1, then $\varphi(x) = (1, x_1, x_2, ..., x_D)^T$.
  - If the degree is 2, then $\varphi(x) = (1, x_1, (x_1)^2, x_2, (x_2)^2..., x_D, (x_D)^2)^T$.
  - If the degree is 3, then $\varphi(x) = (1, x_1, (x_1)^2, (x_1)^3, x_2, (x_2)^2, (x_2)^3, ..., x_D, (x_D)^2, (x_D)^3)^T$.
- The fourth argument, `lambda`, is a non-negative real number (it can be zero or greater than zero). This is the value of λ that you should use for regularization. If $\lambda = 0$, then no regularization is used.

The training and test files will follow the same format as the text files in the UCI datasets directory. A description of the datasets and the file format can be found on this link.

However, the datasets in the UCI datasets directory are not the best fits for this program. While you can certainly apply the program and get results, those datasets describe classification problems as opposed to regression problems. A popular regression dataset that you can test your code with is the Boston Housing dataset. You can download the Boston Housing files in the same format that we use for the UCI datasets from these links:

- boston_housing_training.txt: The training file, with 404 rows corresponding to 404 training examples, and 14 columns. As usual, the first 13 columns are the observations that are provided as input to the system, and the 14th column is the target output.
- boston_housing_test.txt: The test file, with 102 rows corresponding to 102 test examples, and 14 columns (again, 13-dimensional inputs followed by the target output).

One note: compared to the original Boston Housing dataset, the data in our Boston Housing files have been normalized (by dividing by a factor of 711) to make sure that each attribute value is between 0 and 1.

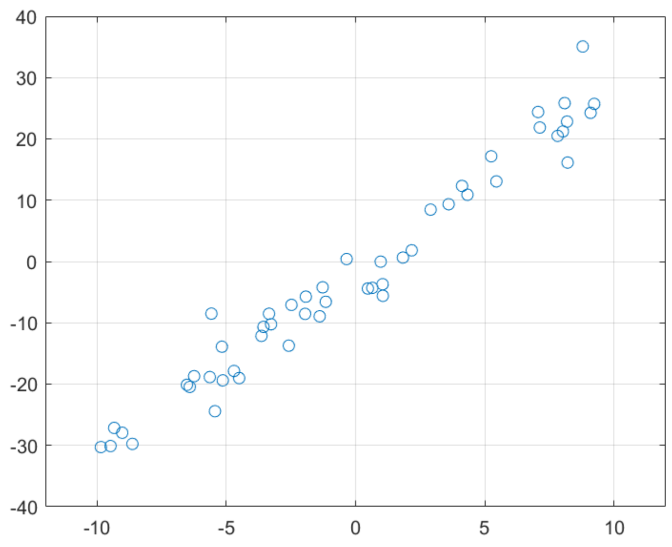
Your code should also work with ANY OTHER training and test files using the same format.

As a reminder, do NOT use data from the last column (i.e., the target outputs) as features.

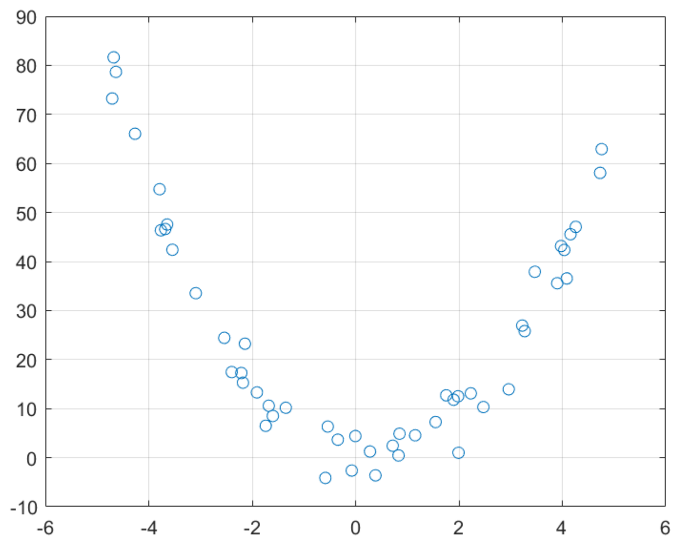
1-Dimensional Datasets

To verify and debug your code, you can also use the following 1-dimensional datasets (first column is input, second column is target output). For each dataset, you can also see a plot, where the input corresponds to the x axis and the target output corresponds to the y axis. These datasets were generated randomly, using the Matlab code in script.m, f1.m, f2.m, and f3.m.

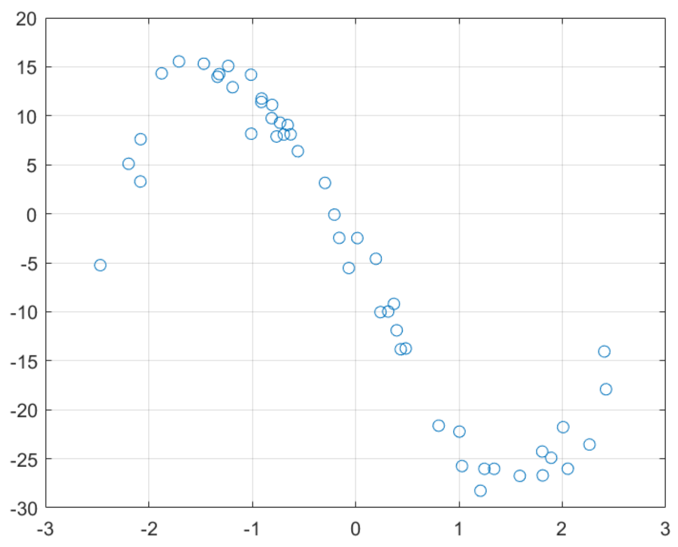
- f1.txt: generated using function t(x) = 3x - 2 + noise.



- f2.txt: generated using function t(x) = $3x^2$ - 2x + 1 + noise.



- f3.txt: generated using function $t(x) = 3x^3 - x^2 - 20x - 4 + \text{noise}$.



Training Stage

Remember to use the pseudo-inverse function in Python (numpy.linalg.pinv) when you need to invert matrices.

At the end of the training stage, your program should print out the values of the weights that you have estimated. The output of the training phase should be a sequence of lines like this:

```
w0=%.4f
w1=%.4f
w2=%.4f
...
```

Test Stage

After the training stage, you should apply the function that you have learned on the test data. For each test object (following the order in which each test object appears in the test file), you should print a line containing the following info:

- object ID. This is the line number where that object occurs in the test file. Start with 1 in numbering the objects, not with 0.
- output of the function that you have learned for that object.
- target value (the last column on the line where the object occurs).
- squared error. This is simply the squared difference between the output that your function produces for the test object and the target output for that object.

The output of the test stage should be a sequence of lines like this:

```
ID=%5d, output=%14.4f, target value = %10.4f, squared error = %.4f
```

Object IDs should be numbered starting from 1, not 0. Lines should appear sorted in increasing order of object ID.

In your answers.pdf document, provide the full output of the training stage, and ONLY THE LAST LINE (the line printing the result on the last test object) of the output by the test stage of your program, when given boston_housing_training.txt as the training file, and boston_housing_test.txt as the test file. Provide this output for all four combinations where degree=1 or degree=2 (where degree is the second argument) and λ=0 or λ=1.

Example Output

Students usually ask for an example of correct output for this program. In the real world, usually you will not be able to obtain such examples. To best prepare for the real world, I strongly recommend that you try to develop and test your solution without using such an example output. Nonetheless, if you decide that you need to see examples of output, you can use these examples:

- f1_1_0.txt. Output of `linear_regression('f1.txt', 'f1.txt', 1, 0)`
- f3_3_0.txt. Output of `linear_regression('f3.txt', 'f3.txt', 3, 0)`
- f3_3_1.txt. Output of `linear_regression('f3.txt', 'f3.txt', 3, 1)`
- boston_2_0.txt. Output of `linear_regression('boston_housing_training.txt', 'boston_housing_test.txt', 2, 0)`
- boston_2_1.txt. Output of `linear_regression('boston_housing_training.txt', 'boston_housing_test.txt', 2, 1)`

Task 2 (15 points, written)

We are given these training examples for a linear regression problem:

$x_1 = 5.3$, $t_1 = 9.6$
$x_2 = 7.1$, $t_2 = 4.2$
$x_3 = 6.4$, $t_3 = 2.2$

We want to fit a line to this data, so we want to find the 2-dimensional vector **w** that minimizes $\tilde{E}_D(\mathbf{w})$ as defined in slide 60 of the linear regression slides. What is the value of **w** in the limit where λ approaches positive infinity? Justify your answer. Correct answers with insufficient justification will not receive credit.

Task 3 (15 points, written)

We are given these training examples for a linear regression problem:

$x_1 = 5.3$, $t_1 = 9.6$
$x_2 = 7.1$, $t_2 = 4.2$

$x_3 = 6.4$, $t_3 = 2.2$

We are also given these two lines as possible solutions:

- `f(x) = 3.1x + 4.2`
- `f(x) = 2.4x - 1.5`

Which of these lines is a better solution according to the sum-of-squares criterion? This criterion is defined as function $E_D(\mathbf{w})$ in slide 25 of the linear regression slides. Justify your answer. Correct answers with insufficient justification will not receive credit.

Task 4 (10 points, written)

Your colleague, Bob Hacker, has come up with a new (and better, he claims) training algorithm for the regularized version of linear regression. Bob's algorithm, instead of requiring lambda as an input, computes both the optimal w and the optimal lambda that minimize $\tilde{E}_D(\mathbf{w})$ as defined in slide 60 of the linear regression slides. Bob says that his algorithm is better, since it does not require lambda as a hyperparameter, and instead computes the best lambda automatically (together with the best w).

Your boss is asking for your opinion: should your company use Bob's algorithm instead of the standard training algorithm that you were asked to implement in Task 1? First you check Bob's algorithm, and you verify that indeed it works correctly, and computes both the optimal w and the optimal lambda. Based on that, what is your recommendation? Explain as specifically as possible why Bob's algorithm should or should not replace the standard algorithm.
