COMPSCI 590N: Assignment 4

Due: October 9, 2017 at 11:55pm

Included with the assignment is a script for testing your solution called `assignment4_tests.py`. This script will test the output from your code against a number of test cases and will indicate if there are errors. Once you have written your code in `assignment4.py`, you can run these tests by executing:

```
python assignment4_tests.py
```

Be sure that you can run `assignment4_tests.py` before submitting as this is how we will test your code for grading! The provided test cases are meant to help you debug your code, but you should not assume that they are exhaustive. If a problem asks you define a function or class, **you should use exactly the name specified** in the problem for this function or class. Your modified version of `assignment4.py` should be submitted to Moodle by the due date specified above.

Please submit to Moodle a zip file containing your code file as well as a PDF containing all plots and question responses.

1 Problem 1: Gauss-Jordan Elimination

Please write a function called `gauss_jordan` that implements matrix inversion using Gauss-Jordan elimination, described in pseudo-code in Algorithm 1. Your function should take as input a square NumPy array, A, and should return A^{-1} if A is invertible and `None` if it is not. You may assume that the input will be a square, two dimensional NumPy array with a numeric `dtype`.

2 Problem 2: Ordinary Least Squares Linear Regression

Linear regression is a common model for identifying linear relationships between a dependent variable, y, and a set of dependent variables, $\mathbf{x}$. Linear regression assumes the following model,

$$y = \beta^T \mathbf{x} + \epsilon$$

Algorithm 1 Gauss-Jordan Elimination

```
for each row k do
  $i^* \leftarrow \arg\max_{k \le i \le n} |A_{ik}|$
  if $A_{i^*k} = 0$ then
    Matrix is not invertible
  end if
  Swap rows $k$ and $i^*$
  for each row $j$ below $k$ (i.e. $j = k+1, ..., n$) do
    $f = \frac{A_{jk}}{A_{kk}}$
    $A_j = A_j - fA_k$
  end for
end for
for each row $k = n, ..., 1$ (i.e. in reverse) do
  $A_k = A_k / A_{kk}$
  for each row $j$ above $k$ (i.e. $j = k-1, ..., 1$) do
    $f = \frac{A_{jk}}{A_{kk}}$
    $A_j = A_j - fA_k$
  end for
end for
```

where β is a vector coefficients that we would like to estimate and ϵ is Normally distributed noise. Given a set of n data cases of m covariates arranged as a matrix $\mathbf{X} \in \mathbb{R}^{n \times m}$ and a corresponding set of dependent variable observations arranged as a vector $\mathbf{y} \in \mathbb{R}^n$, we can estimate β by minimizing the sum of squared errors,

$$\begin{aligned}\hat{\beta} &= \arg\min_{\beta} \sum_{i=1}^{n} (\beta^T \mathbf{X}_i - \mathbf{y}_i)^2 \\ &= \arg\min_{\beta} (\mathbf{X}\beta - \mathbf{y})^T (\mathbf{X}\beta - \mathbf{y}) \\ &= (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}\end{aligned}$$

In this problem, you will implement and compare two versions of this estimation method. Both methods should take two arguments, a two dimensional array $\mathbf{X}$ containing the independent variable observations and a one dimensional array $\mathbf{y}$ containing the dependent variable observations. Both methods should return a one dimensional vector containing $\hat{\beta}$. The two implementations are as follows,

- `linear_regression_inverse` should estimate $\hat{\beta}$ using only matrix multiplications and matrix inversion.
- The Moore-Penrose pseudo-inverse of a matrix $\mathbf{X}$ is defined as $\mathbf{X}^+ = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T$. `linear_regression_moore_penrose` should estimate $\hat{\beta}$ using the numpy function for finding the Moore-Penrose pseudo-inverse of a matrix, `numpy.linalg.pinv`.

You should use the numpy implementations of matrix multiplication, inversion, and pseudo-inversion. Once you have implemented both versions of linear regression, you should compare the runtimes of the two implementations. To do this, you should generate random data of varying sizes and look at how the runtime changes as you vary the input size. Code is provided for generating data and timing your functions. You should generate the following plots and answer the corresponding questions:

1. With the number of data cases, n, fixed at 1000, vary the number of covariates, m, between 25 and 250. Generate a plot where the x-axis is m and the y-axis is the algorithms' runtimes in seconds. Plot the results for both methods on the same axis.

2. With the number of covariates, m, fixed at 25, vary the number of data cases, n, between $1,000$ and $10,000$. Generate a plot where the x-axis is n and the y-axis is the algorithms' runtimes in seconds. Plot the results for both methods on the same axis.

3. Which of the two methods is faster?

4. Which of the two data dimensions, m or n, has a greater impact on the run time? Why is this the case? (Hint: Look at the dimensionality of the intermediate calculations in $(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$)

This problem requires you to generate some simple plots. There are many python plotting modules, however, I highly recommend using `matplotlib` which has become the de facto standard for data analysis tasks. The documentation for the standard plotting function `matplotlib.pyplot.plot` can be found at `http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.plot` and some examples can be found at `http://matplotlib.org/users/pyplot_tutorial.html`.