Assignment 3

Due Date: March 4, 8:59pm

Please see the guidelines at https://www.cs.toronto.edu/~lczhang/338/homework.html

What to Hand In

Please hand in 2 files:

- Python File containing all your code, named `csc338_a3.py`. If you are using Jupyter Notebook to complete the work, your notebook can be exported as a .py file (File -> Download As -> Python). Your code will be auto-graded using Python 3.6, so please make sure that your code runs. There will be a 20% penalty if you need a remark due to small issues that renders your code untestable.
- PDF file named `csc338_a3_written.pdf` containing your solutions to the written parts of the assignment. Your solution can be hand-written, but must be legible. Graders may deduct marks for illegible or poorly presented solutions.

Submit the assignment on **MarkUs** by 9pm on the due date. See the syllabus for the course policy regarding late assignments. All assignments must be done individually.

Q1. Positive Definite Matrices [10 pt]

Part (a) [4 pt]

Are the following matrices symmetric positive definite? Save each answer (`True` or `False`) in the variables `M1_pd`, … `M4_pd`.

You should be able to do this question by hand.

Part (b) [6 pt]

Suppose that A is a symmetric positive definite matrix. Show that the function

$$||x||_A = (x^T A x)^{\frac{1}{2}}$$

on a vector x satisfies the three properties of a vector norm.

Include your solution in your pdf writeup.

Q2. Cholesky Factorization [10 pt]

Write a function `cholesky_factorize` that returns the Cholesky factorization of a matrix, according to its docstring

Q3. Rank-k Update Woodbury Formula [12 pt]

Part (a) [6 pt]

Recall that we proved the Sherman-Morrison formula in class:

$$(A - uv^T)^{-1} = A^{-1} + A^{-1}u(1 - v^T A^{-1}u)^{-1}v^T A^{-1}$$

A related formula is the Woodbury formula for a rank-k update of the matrix A. Specifically, if U and V are $n \times k$ matrices, then

$$(A - UV^T)^{-1} = A^{-1} + A^{-1}U(I - V^T A^{-1}U)^{-1}V^T A^{-1}$$

Prove that the Woodbury formula is correct. Include your solution in your pdf writeup.

Part (b) [6 pt]

A rank 1 matrix has the form xy^T where x and y are column vectors. Suppose A and B are non-sigular matrices. Show that $A - B$ is rank 1 if and only if $A^{-1} - B^{-1}$ is also rank 1.

Include your solution in your pdf writeup.

Hint: Use the Sherman-Morrison formula. This question is not supposed to be easy, so leave aside time to think!

Q4. Iterative Refinement [6 pt]

For this question, we will use the code from assignment 2.

Part (a) [1 pt]

Recall the following system for question 2(d) of assignment 2.

When we tried to solve this system using gauss elimiation without pivoting, we obtained an answer `q4_x0` below that is incorrect. Compute the residual of `q4_x0`. Save the residual in the variable `q4_r0`

Part (b) [4 pt]

We will use iterative refinement to produce a better solution `q4_x1` to the system $Ax = b$. First, what is the new system that we need to solve? Save the A and b of the new system in the variable `q4_A1` and `q4_b1`. Compute the residual `q4_r1` of your new solution.

Part (c) [1 pt]

How does your new solution `q4_x1` compare to your old solution `q4_x0`?

Store either "better" or "worse" in the variable `new_solution_is`, depending on whether you think `q4_x1` is a better or worse solution than `q4_x0` to the initial system $Ax = b$.

Q5. Normal Equation [10 pt]

Part (a) [5 pt]

Write a function `solve_normal_equation` that takes an $m \times n$ matrix A and a vector b, and solves the linear least squares problem $Ax \approx b$ using the Normal Equation method.

You should use the `cholesky_factorize` method that you wrote in question 3. You may need other functions that you wrote in assignment 2.

Part (b) [2 pt]

Consider the linear least square system below, where A is a 50×3 matrix, and b is a vector of size 50. The system is derived from the "iris" dataset.

Use the `solve_normal_equation` function you wrote in part (a) to find the vector `x` that minimizes the norm of `matmul(A, x) - b`.

Store the solution in the variable `q5b_x` and the residual in `q5b_r`. You may use the function `np.linalg.norm`.

Part (c) [1 pt]

Consider an alternative value of x for the system $Ax \approx b$ defined in part (b), with `x = np.array([-0.2, 0.2, 0.5])`. Compute the norm of the residual $Ax - b$ for this alternate value of x, and save the result in the variable `q5c_r`.

You should see that `q5b_r < q5c_r`.

Part (d) [2 pt]

Show that if an $m \times n$ matrix A with $m > n$ has $rank(n)$, then $A^T A$ is positive definite.

Q6. Householder Transforms [17 pt]

Part (a) [2 pt]

Consider the following linear least squares problem. What is the minimum value of $||Ax - b||_2$? Save the minimum norm of the residual in the variable `q5a_r`.

What is the solution to the problem? Save the value in the variable `q5a_x`.

Do this entire problem by hand. You should verify that the residual matches the value you stored in `q5_r`.

Part (b) [5 pt]

Write a function `householder_v` that returns the vector v that defines the Householder transform

$$H = I - 2\frac{vv^T}{v^T v}$$

that eliminates all but the first element of a vector a.

Part (c) [3 pt]

Write a function `apply_householder` that applies the Householder transform defined by a vector v to a vector u. You should **not** compute the Householder transform matrix H. You should only need to compute vector-vector dot products and vector-scalar multiplications.

Part (d) [3 pt]

Write a function `apply_householder_matrix` that applies the Householder transform defined by a vector v to all the columns of a matrix U. You should **not** compute the Householder transform matrix H.

Do not use for loops. Instead, you may find the numpy function `np.outer` useful.

Part (e) [4 pt]

Write a function `solve_qr_householder` that takes an $m \times n$ matrix A and a vector b, and solves the linear least squares problem $Ax \approx b$ using Householder QR Factorization. You may use `np.linalg.solve` to solve any square system of the form $Ax = b$ that you produce.

You should use the helper function `qr_householder` that takes a matrix A and a vector b and performs the Householder QR Factorization using the functions you wrote in parts (b-d).

Part (f) [2 pt]

Use the function `solve_qr_householder` to solve the Iris system from Q5(b).

Save your result for x in the variable `q6f_x`.

Do you get the same result or a different result from Q5(b)? Change the value of the variable `q6f_compare` to either the string "same" or "different" depending on whether your results were the same or different.

Part (g) [4 pt]

Show that for $H = I - 2\frac{vv^T}{v^Tv}$, we have $H = H^T = H^{-1}$.

Include your solution in your PDF writeup.

Q7. MNIST Digit Recognition [15pt]

For the next few questions, we will use the MNIST dataset for digit recognition Download the files `mnist_images.npy` and `mnist_labels.npy` from the course website, and place them into the same folder as your ipynb file.

The code below loads the data, splits it into "train" and "test" sets, and plots the a subset of the data. We will use `train_images` and `train_labels` to set up Linear Least Squares problems. We will use `test_images` and `test_labels` to test the models that we build.

Part (a) [2 pt]

How many examples of each digit are in `train_images`? You should use the information in `train_labels`.

If you are not well versed in numpy, some of the code in the later part of this question might be helpful. You might also find the `sum` function helpful.

Save your result in the array `mnist_digits`, where `mnist_digits[0]` should contain the number of digit `0` in `train_images`, `mnist_digits[1]` should contain the number of digit `1` in `train_images`, etc.

Part (b) [2 pt]

We will build a rudimentary model to predict whether a digit is the digit `0`. Our features will be the intensity at each pixel. There are 28 * 28 = 784 pixels in each image. However, in order to obtain a matrix A that is of full rank, we will ignore the pixels along the border. That is, we will only use 400 pixels in the center of the image.

Look at a few of the MNIST images using the `plot_mnist` function written for you. Why would our matrix A not be full rank if we use all 784 pixels in our model?

If this question doesn't make sense yet, you might want to come back to it after completing the rest of this question.

Include your solution in your PDF writeup.

Part (c) [1 pt]

We will now build a rudimentary model to predict whether a digit is the digit `0`. To obtain a matrix of full rank, we will use the 400 pixels at the center of each image as features. Our target will be whether our digit is `0`.

In short, the model we will build looks like this:

$$x_1 p_1 + x_2 p_2 + ... + x_{400} p_{400} = y$$

Where p_i is the pixel intensity at pixel i (the ordering of the pixel's doesn't actually matter), and the value of y determines whether our digit is a `0` or not.

We will solve for the coefficients x_i by solving the linear least squares problem $Ax \approx b$, where A is constructed using the pixel intensities of the images in `train_images`, and y is constructed using the labels for those images. For convenience, we will set $y = 1$ for images of the digit `0`, and $y = 0$ for other digits.

I should stress that in CSC411, you will learn that this is not the proper way to build a digit detector. However, digit detection is quite fun, so we might as well use to tools that we have to try and solve the problem.

The code below obtains the matrices A and the vector b of our least squares problem, where A is a $m \times n$ matrix and b is a vector of length m.

What is the value of m and n? Save the values in the variables `mnist_m` and `mnist_n`.

Part (d) [2 pt]

Use the Normal Equations method to solve the system $Ax = b$. Save the result in the variable `mnist_x1`. Save the norm of the residuals of this solution in the variable `mnist_r1`.

Then, use the Householder QR decomposition method to solve the same system. Save the result in the variable `mnist_x2`. Save the norm of the residuals of this solution in the variable `mnist_r2`.

Part (e) [1 pt]

Consider `test_images[0]`. Is this image of the digit 0? Set the value of `test_image_0` to either `True` or `False` depending on your result.

Let p be the vector containing the values of the 400 center pixels in `test_image[0]`. The features are extracted for you in the variabel `p`. Use the solution `mnist_x1` to estimate the target y for the vector p. Save the (float) value of the predicted value of y in the variable `test_image_0_y`.

Part (f) [2 pt]

Write code to predict the value of y for **every** image in `test_images`. Save your result in `test_image_y`.

Do not use a loop.

Part (g) [2 pt]

We will want to turn the continuous estimates of y into discrete predictions about whether an image is of the digit 0.

We will do this by selecting a **cutoff**. That is, we will predict that a test image is of the digit 0 if the prediction y for that digit is at least 0.5.

Create a numpy array `test_image_pred` with `test_image_pred[i] == 1` if `test_image[i]` is of the digit 0, and `test_image_pred[i] == 0` otherwise. Then, run the code to compute the `test_accuracy`, or the portion of the times that our prediction matches the actual label.

HINT: You might find the code in Part(c) helpful.

(This is somewhat of an arbitrary cutoff. You will learn the proper way to do this prediction problem in CSC411.)

Part (h) [3 pt]

So far, we built a linear least squares model that determines whether an image is of the digit 0. Let's go a step further, and build such a model for **every** digit!

Complete the function `mnist_classifiers` that uses `train_images` and `train_labels`, and uses the function `solve_normal_equation` to build a linear least squares model for every digit. The function should return a matrix xs of shape 10×400, with `xs[0] == mnist_x1` from earlier.

Please comment out any code you use to test `mnist_classifier`. It will help make testing your code faster, so I can get your assignments back to you more quickly.

Just for fun...

The code below makes predictions based on the result of your `mnist_classifier`. That is, for every test image, the code runs all 10 models to see whether the test image contains each of the 10 digits. We make a discrete prediction about which digit the image contains by looking at which model yields the **largest** value of y for the image.

The code then compares the result against the actual labels, computes the accuracy measure: the fraction of predictions that is correct. Just for fun, look at the prediction accuracy of our model, but please comment any code you write before submitting your assignment.

Again, in CSC411 you will learn much better ways of classifying digits. You will achieve a much better test accuracy than what we have here.