University of Toronto Mississauga
Department of Mathematical and Computational Sciences
CSC 338 - Numerical Methods, Spring 2017

Assignment 3

Due date: Friday March 31, 11pm.
No late assignments will be accepted.

The material you hand in should be legible (either typed or *neatly* hand-written), well-organized and easy to mark, including the use of good English. In general, short, simple answers are worth more than long, complicated ones. Unless stated otherwise, all answers should be justified.

All computer problems are to be done in Python with NumPy and SciPy and should be properly commented and easy to understand. Hand in all the programs you are asked to write. They should be stored in a single file called `source.py`. We should be able to import this file as a Python module and execute your programs. For generating plots, you may find the SciPy functions `numpy.linspace` and `matplotlib.pyplot.plot` useful. Note that if `A` and `B` are two-dimensional numpy arrays, then `A*B` performs *pointwise* multiplication, *not* matrix multiplication. To perform matrix multiplication, you can use `numpy.dot(A,B)`.

You should hand in four files: the source code of all your Python programs, a pdf file of all the requested program output, answers to all the non-programming questions (scanned hand-writing is fine), and a scanned, signed copy of the cover sheet at the end of the assignment. Be sure to indicate clearly which question(s) each program and piece of output refers to. The four files should be submitted electronically as described on the course web page.

I don't know policy: If you do not know the answer to a question (or part), and you write "I don't know", you will receive 20% of the marks of that question (or part). If you just leave a question blank with no such statement, you get 0 marks for that question.

No more questions will be added

1. (6 points total) Write out the statement for updating the iterate x_k using Newton's method for solving each of the following equations (2 points each):

   (a) $x^3 + 2x^2 - 3 = 0$

   (b) $x \sin 2x = 1$

   (c) $e^x = e^{2x} - x$

2. (6 points) Starting at $(x_1, x_2) = (1, 1)^T$, carry out one iteration of Newton's method applied to the following system of equations:

$$\begin{aligned} x_1^2 + x_2^2 &= 1 \\ x_1^2 - x_2^2 &= 0 \end{aligned}$$

3. (15 points total) We wish to find the cube root of a number, y.

   (a) (3 points) What is the condition number for this problem? For what values of y, if any, is the condition number at most 1?

   (b) (4 points) How can we use Newton's method to solve this problem? Write down the statement for updating x_k.

   Show that finding the cube root of y is equivalent to finding the fixed point of the functions, g_i, below. In each case, determine for what values of y, if any, the fixed-point iteration is locally convergent.

   (c) (4 points) $g_1(x) = x^3/y + x - 1$

   (d) (4 points) $g_2(x) = y/x^2$

4. (11 points total) Suppose we replace the derivative $f'(x_k)$ in Newton's method by a constant, s, so that the iterative update step becomes

$$x_{k+1} = x_k - f(x_k)/s$$

   (a) (4 points) Under what conditions on the value of s will this method be locally convergent? Simplify your results.

   (b) (3 points) In general, what will be the convergence rate, r, and the convergence constant, C.

   (c) (4 points) For what value(s) of s, if any, will the convergence be quadratic?

5. (11 points total) Consider the following function:

$$f(x, y) = x^4 - 4xy + y^4 - 3$$

   (a) (9 points) Determine the critical points of the function, and characterize each as a maximum, minimum or saddle point. Hint: see pages 263 and 264 in the text, especially the comment on Cholesky factorization. Also note that a matrix, A, is negative-definite if and only if $-A$ is positive definite.

(b) (2 points) Write out the statement for updating the iterate (x_k, y_k) using Newtons method for optimization.

6. (4 points) We wish to determine the value, x, at which a univariate function $f(x)$ achieves its minimum. Suppose we can do this to within 7 digits of accuracy by using just the values of the function. To roughly how many digits of accuracy can we determine the minimum by using the derivatives of the function? i.e, by solving the equation $f'(x) = 0$.

Optical Character Recognition, Part III.

In the rest of this assignment, you will use your programs from Assignments 1 and 2 to train and test a program that recognizes hand-written digits. If you did not finish the programs, you may use the programs provided in the solutions to Assignments 1 and 2. You may also use someone else's programs *with their permission*. In any case, be sure to clearly state whose programs you are using. Whatever programs you use, you should test them to make sure they work properly (as in Questions 10(b) and (c) of Assignment 2), as *you will be responsible* for any errors they make.

Please re-read the description of the MNIST data in Assignment 1. Recall that it consists of images of hand-written digits, where each digit has $28 \times 28 = 784$ pixels. There are two sets of images, 60,000 for training and 10,000 for testing. Each set is stored as a tuple, $(X_0, X_1, ..., X_9)$, where each X_d is a matrix, and each row of X_d is an image of digit d. X_d is thus a $m_d \times n$ matrix, where m_d is the number of images of digit d, and n is the number of pixels per image (784).

One thing to be careful of below is the overloaded use of the symbol Σ. It is common in the field for Σ to represent a covariance matrix. But, of course, it can also represent summation. The meaning will usually be clear from context, but also, this assignment generally uses a larger Σ to represent summation. For example,

$$\sum_i x_i$$

means the sum of all the x_i.

As in Assignments 1 and 2, your Python programs should minimize the use of loops. Instead, use Numpy's vector and matrix operations, which are much faster and can be executed in parallel. In fact, for full marks, you should not use any doubly-nested loops, and any loops you do use should only iterate over digits, that is, they should only iterate from 0 to 9.

Also note that you evaluate your programs in Question 9. If you do not do this question, then you have no way of knowing if your programs are correct. In this case, we may run your programs outselves. If we cannot run them or if they produce errors or absurd results, then *you may receive 0* for them.

7. (20 points total) *Training.*

The first step is to use your programs from Assignment 2 to learn ten multivariate normal distributions, one for each digit. That is, for each matrix X_d in the training data, you will estimate a mean vector, μ_d, and a covariance matrix, Σ_d. However, unlike Assignment 2, μ_d now has 784 dimensions, and Σ_d is a 784×784 matrix. Each multivariate normal thus has $784 + 784^2 = 615,400$ parameters, for a total of 6,154,000 parameters for all ten digits. Unfortunately, we cannot possibly learn this many parameters, since we have only 60,000 training samples, and one typically needs at least 5-10 training samples per parameter to get a good estimate of parameter values.

This is a common problem in Machine Learning, called *overfitting*. To solve it, we will fit two different models to each digit, the multivariate normal model described above and a simpler model with far fewer parameters. We call these the complex and simple models, respectively. We will combine these two models to get a model of intermediate complexity, one that is good at making predictions. (This idea is called *regularization*.)

Like the complex model, the simple model uses a multivariate normal for each digit. It also uses the same mean vectors, μ_d. However, the ten covariance matrices are all replaced by a single matrix, $I\sigma^2$, where I is the identity matrix. Thus, the 10×784^2 covariance parameters of the complex model are replaced by a single parameter, σ^2, which represents the average variance of the pixels. Part (a) below discusses how to estimate the value of σ^2.

To combine the complex and simple models, we again use multivariate normals with mean vectors μ_d. However, we use a weighted average of the covariance matrices of the complex and simple models. That is, in the combined model, the covariance matrix of digit d is

$$\overline{\Sigma}_d = \beta\Sigma_d + (1-\beta)I\sigma^2 \tag{1}$$

where $0 \leq \beta \leq 1$. Intuitively, β is the weight given to the complex model, and $1-\beta$ is the weight given to the simple model. One of your tasks in Question 9 will be to choose a good value for β.

In this question, your task is to compute values for μ_d, Σ_d and σ^2. We break this into two parts. In the first part, you derive a convenient formula for computing σ^2.

(a) (8 points) The parameter σ^2 represents the average variance of the pixels. That is,

$$\sigma^2 \;=\; \sum_{ijd}(x_{ijd} - \mu_{jd})^2/nm$$

Here, x_{ijd} is the ij^{th} component of matrix X_d, and μ_{jd} is the j^{th} component of vector μ_d. Also, n is the number of columns in X_d, and m is the total number of images. That is, $m = m_0 + ... + m_9$, where m_d is the number of rows in X_d.

Since we are going to compute the Σ_d, we can use them to simplify the computation of σ^2. In particular, show that

$$\sigma^2 \;=\; \sum_{d} m_d\sigma_d^2/m \tag{2}$$

where σ_d^2 is the average value of the diagonal elements of matrix Σ_d. Notice that we now have a single sum (over d) instead of a triple sum (over i, j and d). This is much easier to compute. In proving Equation (2), recall from Assignment 2 that the kj^{th} entry of matrix Σ_d is given by

$$\Sigma_{kjd} = \sum_i (x_{ikd} - \mu_{kd})(x_{ijd} - \mu_{jd})/m_d$$

where the sum is over the rows of matrix X_d.

(b) (12 points) Write a Python function `train(data)` that learns σ^2, μ_d and Σ_d for all d. The argument, `data`, is a set of training data similar in form to the MNIST training data described above. Your function should work for any problem with training data of this form. For example, in addition to hand-written digits, which have 10 classes, it should also work given training data for handwritten letters, which have 26 classes. Your program should also work if the images are not 28×28, though you may assume they are square.

Specifically, `data` is a tuple $(X_0, X_1, ..., X_D)$, where each X_d is a data matrix for class d, and the number of classes is $D + 1$. The `train` function should return a triple `(mu,Sigma,var)`, where `mu` is the list $[\mu_0, ..., \mu_D]$, `Sigma` is the list $[\Sigma_0, ..., \Sigma_D]$, and `var` (which stands for variance) is the real number σ^2. Use the function `fitNormal` from Assignment 2 to compute each μ_d and Σ_d, and use equation (2) above to compute σ^2.

8. (25 points total) *Prediction.*

In Question 7, you used the MNIST training data to learn ten probabilistic models, one for each digit. The next step is to use them to make predictions on the MNIST test data. In this question, you will carry out the computationally most intensive part of the process. For each test image, x_i, and each digit, d, you will compute the probability, p_d, of the image under the model for digit d. Thus, for each test image, we generate a vector $(p_0, ..., p_9)$. This vector can be viewed as a "soft" prediction. That is, we are not committing to a particular prediction, instead, p_d is the likelihood that the image is of digit d. This will form the basis of "hard" predictions in Question 9.

As in Question 7(b), your functions should not be limited to MNIST data. That is, you should not assume there are only ten data matrices, nor that they have 784 columns each.

(a) (5 points) Like the training data, the MNIST test data is a tuple of the form $(X_0, X_1, ..., X_9)$, where X_d is a data matrix for digit d. The first step is to put the test data into a more convenient form for making and evaluating predictions. Write a Python function `flatten(data)` where `data` is a tuple of test data. The function should return a pair (X, Y), where X is a single matrix containing all the test data, and Y is a vector indicating the class of each test point. Specifically, each row of X comes from some matrix X_d in `data`. Thus, if X_d has m_d rows and n columns, then X has m rows and n columns, where $m = m_0 + ... + m_9$. In addition, if x_i is the i^{th} row vector of matrix X, and x_i is an image of digit d

(that is, it comes from matrix X_d), then the i^{th} entry of vector Y is the integer d. Y thus has one entry for each test image and is a m-dimensional vector. You may find it helpful to review the solution to Question 11(b) in Assignment 1.

(b) (5 points) The next step is to use Equation (1) to combine the complex and simple models. Specifically, write a Python function `combine(Sigma,var,beta)` where `Sigma` is a list of covariance matrices, $[\Sigma_0, ..., \Sigma_9]$, `var` is a positive number, and `beta` is a number between 0 and 1. The function should compute a list of new covariance matrices, $[\overline{\Sigma}_0, ..., \overline{\Sigma}_9]$ computed according to Equation (1), where the value of σ^2 is `var`, and the value of β is `beta`.

(c) (5 points) The final step is to compute a "soft" prediction, $(p_0, ..., p_9)$, for each test image, as described above. Here, p_d is the probability of the test image under the model for digit d. Unfortunately, for the MNIST data, these probabilities are so small that they result in numerical underflow.

To see why, recall from Assignment 2 that each digit is modelled as a multivariate normal distribution. The probability of an image, x, is given by

$$p \;=\; \frac{exp[-(x-\mu)^T\Sigma^{-1}(x-\mu)/2]}{(2\pi)^{n/2}\,|\Sigma|^{1/2}}$$

where μ is the mean of the distribution and Σ is the covariance matrix. The problem is that the denominator is so large that it results in overflow. There are two reasons for this. First, the term $(2\pi)^{n/2}$ is huge when $n = 784$ and causes an overflow. Second, $|\Sigma|$, the determinant of Σ, is also huge and causes another overflow. Fortunately, there is a standard fix for such problems: instead of computing probabilities, we shall compute log-probabilities. That is, instead of computing p, we shall compute $\log(p)$, and we shall do this without computing p first. Using the properties of logarithms, it is easy to show that

$$\log(p) \;=\; -\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu) - \frac{n}{2}\log(2\pi) - \frac{1}{2}\log(|\Sigma|) \qquad (3)$$

Notice that the term $(2\pi)^{n/2}$ has disappeared and no longer needs to be computed. The determinant, $|\Sigma|$, is still present, but `numpy` contains functions that will compute the log of a determinant directly from a matrix, without computing the determinant itself first.

Your job in this question is to modify the function `MVNchol` from Question 9(d) in Assignment 2 to return log-probabilities instead of probabilities. You should call this function `logMVNchol`. (If you did not write `MVNchol`, you may modify `MVNinv` instead and call the new function `logMVNinv`.) You should use Equation (3) to compute the log-probabilities, and the function `slogdet` in `numpy.linalg` to compute $\log(|\Sigma|)$.

(d) (5 points) Write a Python function `predict(X,mu,Sigma)` that returns a matrix of log-probabilities, $logP$. Here, `X` is a data matrix, `mu` is a list of mean vectors, $[\mu_0, ..., \mu_9]$, and `Sigma` is a list of covariance matrices, $[\Sigma_0, ..., \Sigma_9]$. μ_d and Σ_d represent the distribution for digit d. For each digit, d, you should use the function

`logMVNchol` to compute a vector of log-probabilities. This vector is column d of $logP$.

Intuitively, this function computes the logarithm of the "soft" predictions discussed above. That is, each row of `X` represents an image, and the corresponding row of $logP$ is a vector of log-probabilities, $[\log(p_0), ..., \log(p_9)]$, where p_d is the probability of the image under the distribution for digit d.

(e) (5 points) Write a Python script that combines all the above functions into a single procedure that uses the output of the `train` function in Question 7(b) to compute the matrix $logP$. The `train` function provides mean vectors μ_d, covariance matrices Σ_d, and a variance σ^2. These will provide the input to your functions in this question. Your script should do the following:

i. Apply the `flatten` function to the MNIST test data to give a single matrix of test data, `X`, and a vector of correct answers, `Y`.

ii. Use the `combine` function to compute the new covariance matrices, $\overline{\Sigma}_d$. (You will need to provide a value for the argument `beta`. For now, use the value 0.5.)

iii. Pass the flattened test data, `X`, the new covariance matrices, $\overline{\Sigma}_d$, and the mean vectors, μ_d, to the `predict` function to compute a matrix of log-probabilities, $logP$.

The script should be handed in as part of the function `ocr()` in Question 9(c).

9. (15 points total) *Evaluation.*

In this question, you will evaluate the predictions made by your functions in Question 8. You will first convert the "soft" predictions returned by the function `predict` into "hard" predictions. You will then compare these to the correct answers, `Y`, returned by the function `flatten`. You will also choose a value for the parameter β in Equation (1), one that maximizes the number of correct predictions.

(a) (10 points) Write a Python function `evaluate(logP,Y)`, where `logP` is a matrix of log-probabilities, and `Y` is a vector of digits. Intuitively, each row of `logP` is a soft prediction, and the corresponding row of `Y` is the correct answer. Your function should do the following:

i. Convert the matrix of soft predictions into a vector, `Yhat`, of hard predictions. For each row of `logP`, the hard prediction is the digit with the highest probability. For instance, if $(logP_0, ..., logP_9)$ is the i^{th} row of `logP` and its largest entry is $logP_6$, then 6 is the hard prediction and becomes the i^{th} entry of `Yhat`.

ii. Compare `Yhat` with `Y` to determine which predictions are correct. The i^{th} entry of `Yhat` is a correct prediction if it is equal to the i^{th} entry of `Y`. Otherwise, it is a false prediction.

iii. Compute and print out the percentage of predictions that are correct.

iv. Choose 36 correct predictions at random (all distinct), and display their images using the function `showImages` from Question 11(a) in Assignment 1. Title the figure, "Some correctly classified images".

v. Choose 36 incorrect predictions at random (all distinct), and display their images using the function `showImages`. Title the figure, "Some misclassified images". You should find that these images are generally not as well-written as the correctly classified images.

You may find it helpful to review the solution to Question 11(b) in Assignment 1. You may also find the `numpy` function `argmax` useful. You can avoid loops by using Numpy boolean index arrays (https://docs.scipy.org/doc/numpy/user/basics.indexing.html).

(b) (5 points) Add a call to `evaluate(logP,Y)` to the end of your Python script in Question 8(c). The value of `logP` should be the output of the `predict` function, and the value of `Y` should be from the output to the `flatten` function.

Run this script using different values for the argument `beta` of the `combine` function. Remember that `beta` is between 0 and 1. You should find that with `beta` = 0 the number of correct predictions is about 82%. With `beta` = 1, you may find you get an error message. This is because the covariance matrices, Σ_d, are singular. (Adding $I\sigma^2$ to them in Equation (1) makes them non-singular.)

You should be able to find a value of `beta` that achieves a prediction accuracy of at least 96%. In any case, report the value of `beta` that gives the highest accuracy. Report the accuracy (percentage of correct predictions) as well. Finally, hand in the two figures generated by the `evaluate` function for this value of `beta`. The script itself should be handed in as part of the function `ocr()` is part (c).

(c) Define a Python function `ocr()` that combines training, prediction and evaluation into a single function that we can run. It should read the MNIST data from a file called `mnist.pickle`, run the function `train` on the training data, and then execute the code from your script in part (b). It should use the value of `beta` that maximizes prediction accuracy. When we run `ocr`, it should produce all the output that you are handing in for parts (a) and (b). You should hand in this function instead of the scripts in Questions 8(e) and 9(b).

University of Toronto Mississauga
CSC 338 - Numerical Methods

Cover sheet for Assignment 3

Complete this page and hand it in with your assignment.

Name: Sayantan Chattopadhyay
(Underline your last name)

Student number: 1000818145

I declare that the solutions to Assignment 1 that I have handed in are solely my own work, and they are in accordance with the University of Toronto Code of Behavior on Academic Matters.

Signature: