University of Toronto Mississauga
Department of Mathematical and Computational Sciences
CSC 411 - Machine Learning and Data Mining, Fall 2018

Assignment 2

Due date: Friday November 15, 11:59pm.
No late assignments will be accepted.

As in all work in this course, 20% of your grade is for quality of presentation, including the use of good English, properly commented and easy-to-understand programs, and clear proofs. In general, short, simple answers are worth more than long, complicated ones. Unless stated otherwise, all answers should be justified. The TA has a limited amount of time to devote to each assignment, so what you hand in should be legible (either typed or *neatly* hand-written), well-organized and easy to evaluate. (An employer would demand no less.) All computer problems are to be done in Python with the NumPy, SciPy and scikit-learn libraries.

Hand in five files: The source code of all your programs (functions and script) in a single Python file, a pdf file of figures generated by the progams, a pdf file of all printed output, a pdf file of answers to all the non-programming questions (scanned hand-writing is fine, but *not* photographs), 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. All the Python code (functions and script) for a given question should appear in one location in your source file, along with a comment giving the question number. All material in all files should appear in order; *i.e.*, material for Question 1 before Question 2 before Question 3, etc. It should be easy for the TA to identify the material for each question. In particular, all figures should be titled, and all printed output should be identified with the Question number. The five files should be submitted electronically as described on the course web page. In addition, if we run your source file, it should not produce any errors, it should produce all the output that you hand in (figures and print outs), and it should be clear which question each piece of output refers to. *Output that is not labled with the Question number will not be graded. Programs that do not produce output will not be graded.*

Style: Use the solutions to Assignment 1 as a guide/model for how to present your proofs, programs and other solutions in this assignment.

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.

Tips on Scientific Programming in Python

If you haven't already done so, please read the NumPy tutorial on the course web page. Be sure to read about indexing and slicing Numpy arrays. First, indexing begins at 0, not 1. Thus, if `A` is a matrix, then `A[7,0]` is the element in row 7 and column 0. Likewise, `A[0,4]` is the element in row 0 and column 4. Slicing allows large segments of an array to be referenced. For example, `A[:,5]` returns column 5 of matrix `A`, and `A[7,[3,6,8]]` returns elements 3, 6 and 8 of row 7. Similarly, if `v` is a vector, then the statement `A[6,:]=v` copies `v` into row 6 of matrix `A`. Note that if `A` and `B` are two-dimensional Numpy arrays, then `A*B` performs *element-wise* multiplication, *not* matrix multiplication. To perform matrix multiplication, you should use `numpy.matmul(A,B)`. Whenever possible, *do not use loops*, which are very slow in Python. In particular, avoid iterating over the elements of a large vector or matrix. Instead, use numpy's vector and matrix operations, which are much faster and can be executed in parallel. For example, if `A` is a matrix and `v` is a column vector, the `A+v` will add `v` to every column of `A`. Likewise for rows and row vectors. Also, the functions `sum` and `mean` in `numpy` are useful for summing or averaging over all or part of an array. Many NumPy functions that are defined for single numbers can be passed lists, vectors and matrices instead. For example, $f([x_1, x_2, ..., x_n])$ returns the list $[f(x_1), f(x_2), ..., f(x_n)]$. The same is true for many user-defined functions. The term `numpy.inf` represents infinity. It results from dividing by 0 in numpy. It can also result from overflow (*i.e.*, from numbers that are too large to represent in the computer, like 10^{1000}). The term `numpy.nan` stands for "not a number", and it results from doing 0/0, inf/inf or inf-inf in numpy. For generating and labelling plots, the following SciPy functions in `matplotlib.pyplot` are needed: `plot`, `xlabel`, `ylabel`, `title`, `suptitle` and `figure`. You can use Google to conveniently look up SciPy functions. e.g., you can google "numpy matmul" and "pyplot suptitle".

Because they are very slow, you should avoid the use of loops and iteration in your Python programs, replacing them by Numpy matrix and vector operations wherever possible. For the same reason, you should not use recursion or higher-order functions (such as the python `map` function or any numpy function listed under "functional programming", such as `apply-along-axis`, which are just just loops in disguise), unless otherwise specified.

1. (? points) *Data Generation.*

   In this question you will generate and plot 2-dimensional data for a binary classification problem. We will call the two classes Class 0 and Class 1 (for which the target values are $t = 0$ and $t = 1$, respectively).

   (a) Write a Python function `gen_data(mu0,mu1,cov0,cov1,N0,N1)` that generates two clusters of data, one for each class. The cluster for class 0 has `N0` points, mean `mu0` and covariance `cov0`. The cluster for class 1 has `N1` points, mean `mu1` and covariance `cov1`. Note that `mu0` and `mu1` and all the data points are 2-dimensional vectors, while `cov0` and `cov1` are real numbers. For both clusters, the variance in each dimension is 1.

   The function should return two arrays, `X` and `t`, representing data points and target values, respectively. `X` is a $N \times 2$ dimensional array, where $N = N0 + N1$ and each row of `X` is a data point. `t` is a N-dimensional vector of 0s and 1s. Specifically, `t[i]` is 0 if `X[i]` belongs to class 0, and 1 if it belongs to class 1. The data for the two classes should be distributed randomly in the arrays. In particular, the data for class 0 should not all be in the first half of the arrays, with the data for class 1 in the second half.

   We will model each cluster as a multivariate normal distribution. Recall that the probability density of such a distribution is given by

$$P(x) = \frac{\exp\left[-(x-\mu)^T\Sigma^{-1}(x-\mu)/2\right]}{\sqrt{(2\pi)^k \det \Sigma}}$$

   where μ is the mean (cluster centre), Σ is the covariance matrix, and k is the dimensionaliy of the data (2 in our case). To generate data for a cluster, use the function `multivariate_normal` in `numpy.random`. Use the function `shuffle` in `sklearn.utils` to distribute the data randomly in the arrays. You should be able to write `gen_data` in at most 10 lines of codes with no loops.

   Note that if $x = (x_1, x_2)$ is a 2-dimensional random variable, then the covariance matrix of x is

$$\begin{pmatrix} \sigma_1^2 & \sigma_{12} \\ \sigma_{12} & \sigma_2^2 \end{pmatrix}$$

   where $\sigma_1^2 = var(x_1)$, $\sigma_2^2 = var(x_2)$ and $\sigma_{12} = cov(x_1, x_2)$. For this problem, $var(x_1) = var(x_2) = 1$ for both classes, and $cov(x_1, x_2)$ is `cov0` for class 0, and `cov1` for class 1.

   (b) Use your function from part (a) to generate two clusters with `N0` = 10,000, `N1` = 5,000, `mu0` = $(1, 1)$, `mu1` = $(2, 2)$, `cov0` = 0 and `cov1` = -0.9.

   (c) Display the data from part (b) as a scatter plot, using red dots for points in class 0, and blue dots for points in class 1. Use the function `scatter` in `numpy.pyplot`. Specify a relatively small dot size by using the named argument `s=2`. Use the functions `xlim` and `ylim` to extend the x and y axes from -3 to 6. Adjust the

Figure 1:

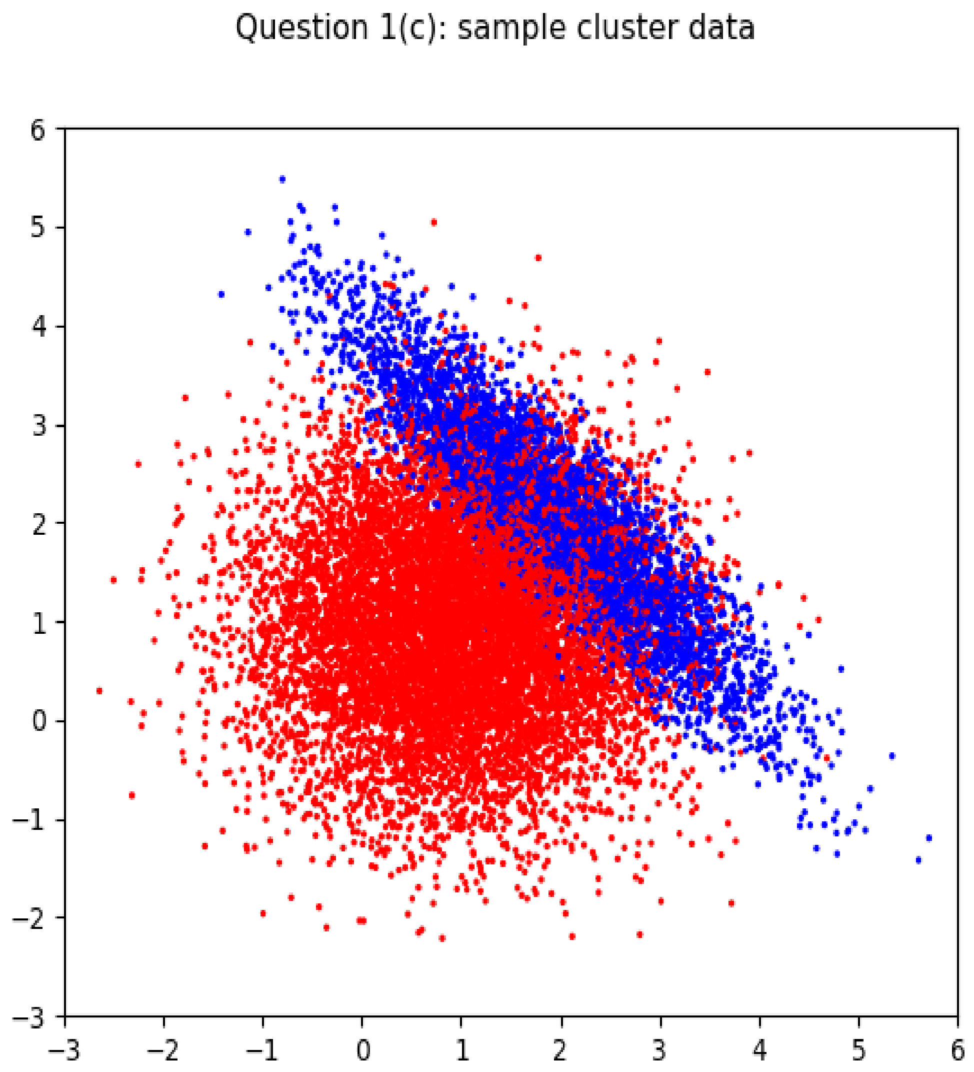


figure window so that the axes look square and the red cluster looks circular. Title the plot, "Question 1(c): sample cluster data".

If you have done everything correctly, the scatter plot should look something like Figure 1, which shows two heavily overlapping clusters. In particular, the red cluster should not be displayed on top of the blue cluster (or vice versa). Instead, the red and blue dots should be intermingled.

2. (? points total) *Binary Logistic Regression.*

In this question you will use logistic regression to generate a classifier for the cluster data. Use the Python class `LogisticRegression` in `sklearn.linear_model` to do the logistic regression. This class generates a Python object, much as the function `Ridge` did in Question 5 of Assignment 1. The class comes with a number of attributes and methods that you will find useful for part (b) below. Do not use any other functions from `sklearn`, and unless specified otherwise, do not use any attributes or methods from `LogisticRegression` except for part (b).

(a) (0 points) Use `gen_data` to generate training data with 1000 points in class 0 and 500 points in class 1. Use the same means and covariances as in Question 1(b).

(b) (? points) Carry out logistic regression on the data in part (a). Print out the values of the bias term, w_0, and the weight vector, w. You can do this in at most 6 lines of code with no loops.

(c) (? points) Compute the accuracy of your logistic-regression classifier on the training data. Do this in two ways: (1) using the `score` method of the `LogisticRegression` class, and (2) from w and w_0 without using any methods of `LogisticRegression` or any functions in `sklearn`. The accuracy is the average number of correct predictions. Call the two estimates of accuracy `accuracy1` and `accuracy2`, respectively. Print out the two estimates of accuracy and their difference. The two estimates should be the same and the difference should be 0. This can all be done in 9 lines of code or less and without loops.

(d) (? points) Generate a scatter plot of the training data as in Question 1(c), and draw the decision boundary of the classifier as a black line on top of the data. Title the figure, "Question 2(d): training data and decision boundary". Do not use any built-in procedures for drawing the decision boundary. Instead, since the decision boundary is a line, you should use the weights in the classifier to draw the line using a single call to `plot`. You can do everything in at most 7 lines of code with no loops.

(e) (? points) Recall that $P(C = 1|x) = 1/(1 + e^{-z})$ where $z = w^T x + w_0$. Generate a scatter plot of the training data, and draw three probability contours on top of the data: $P(C = 1|x) = 0.6$, $P(C = 1|x) = 0.5$ and $P(C = 1|x) = 0.05$. Draw these contours in blue, black and red, respectively. If you have done everything correctly, the black contour should be identical to the decision boundary, the blue contour should pass through a dense region of blue dots, and the red contour should pass through a dense region of red dots. Title the figure, "Question 2(e): three contours". As in part (d), do not use any built-in functions to draw the contours. You can do this in 8 lines of code with no loops if you reuse code from previous questions. In addition to your code, *hand in a written proof of any equations you use.*

(f) (0 points) Use `gen_data` to generate test data with 10,000 points in class 0 and 5,000 points in class 1. Use the same means and covariances as for the training data in part (a).

(g) (? points) Use the test data to compute the precision and recall for each of the three contours in part (e). If you have done everything correctly, the precision should increase as $P(C = 1|x)$ increases, and the recall should decrease. You can do this in at most 16 lines of code with no loops.

3. (? points total) *Generative Models: Theory*

   In the questions below, your proofs should be in the style of those given in the solutions to Assignment 1 (available outside my office door).

   (a) (? points) Prove the second equation on slide 18 of lecture 9:

   $$\phi \; = \; \sum_{n=1}^{N} \mathbb{I}[t^{(n)} = 1]/N$$

   where $\mathbb{I}$ converts true and false to 1 and 0, respectively; *i.e.*, $\mathbb{I}(true) = 1$ and $\mathbb{I}(false) = 0$. (The point is to show that the above equation is the maximum-likelihood solution to the problem stated on slide 17.)

   (b) (? points) Consider the following version of the last equation on slide 18 of Lecture 9:

   $$\Sigma \; = \; \frac{1}{N}\sum_{n=1}^{N}(x^{(n)} - \mu_{t^{(n)}})^T(x^{(n)} - \mu_{t^{(n)}})$$

   where each $x^{(n)}$ is treated as a row vector. Convert this equation to matrix form. That is, let X_0 be the data matrix for class 0, and X_1 be the data matrix for class 1. Each row of a data matrix is an input vector, $x^{(n)}$ (see slide 7 of lecture 9). Prove that

   $$\Sigma \; = \; [Y_0^T Y_0 + Y_1^T Y_1]/N$$

   where $Y_i = X_i - \vec{1}\mu_i^T$, where μ_i is treated as a column vector and $\vec{1}$ is a column vector of 1's.

   (c) (? points) Convert the third and fourth equations on slide 18 to matrix form. That is, derive matrix equations for μ_0 and μ_1.

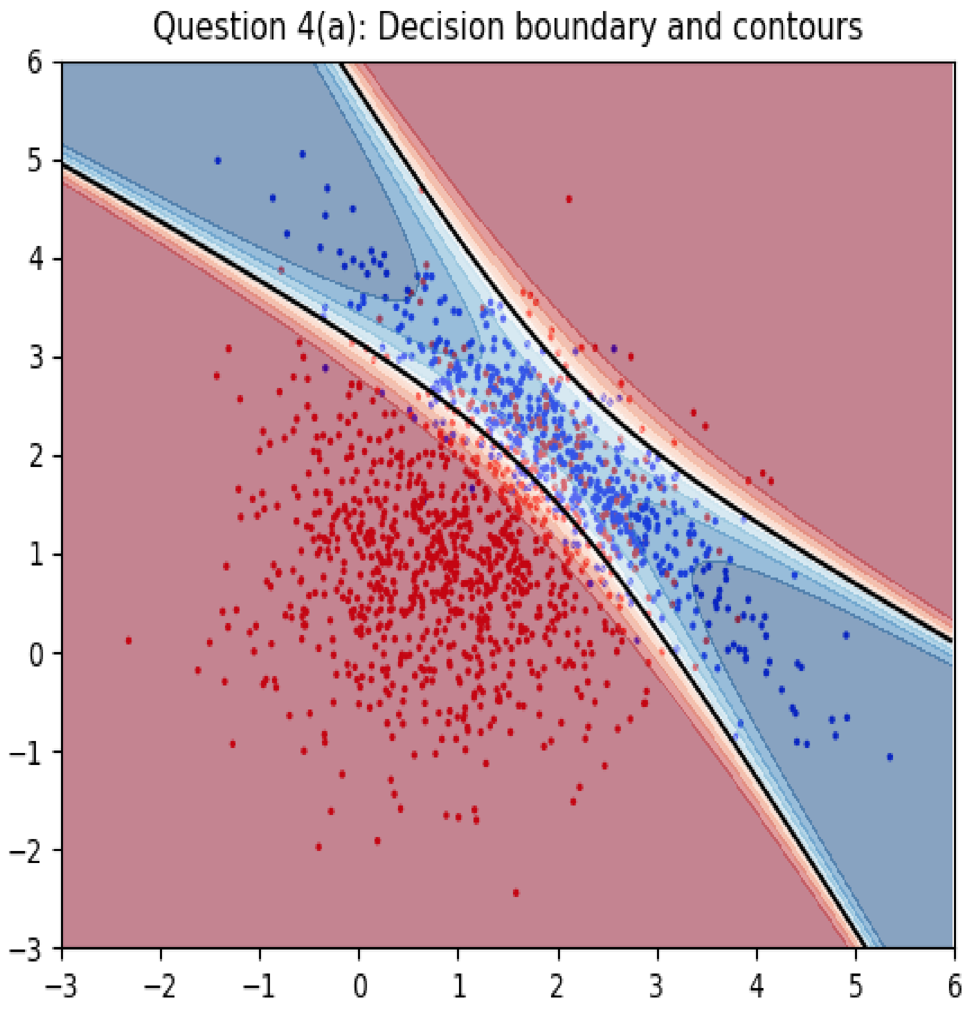
4. (? points) *Generative Models: Practice*

   In this question, you will train a Gaussian Bayes classifier (also known as Gaussian Discriminant Analysis and Quadratic Discriminant Analysis) for the cluster data of Question 2. You will do this in two ways: (*i*) using a method from `sklearn`, and (*ii*) developing your own method. You will also plot decision boundaries using the function `dfContour` from `bonnerlib2`, which you can download from the course web site.[1]

   (a) (? points) Using the Python class `QuadraticDiscriminantAnalysis` in `sklearn.discriminat_analysis` train a Gaussian Bayes classifier on the training data of Question 2. Using the methods and attributes of this class, compute and print out the accuracy of your classifier on the test data of Question 2. In addition, plot the training data, and draw the decision boundary on top of the

[1]`dfContour` plots the decision boundary as a black line and plots several other contours of $P(C = 1|x)$ for a classifier. The colour of a region represents the predicted probability that a point in the region is in class 1 or 0. The darker the blue, the greater the probability that $C = 1$. The darker the red, the greater the probability that $C = 0$.

Figure 2:



training data using `dfContour`. (Be sure to plot the training data *before* you call `dfContour`.) Title the figure, "Question 4(a): Decision boundary and contours" . You can easily do this in 12 lines of code or less, with no loops. If you have done everything correctly, your plot should look something like Figure 2.

(b) As seen in Figure 2, the classifier in part (a) has *two* decision boundaries that define *three* separate regions, two red and one blue. Explain this result. Use diagrams in your explanation.

(c) Generate new training and test data that is the same as that of Question 2 except that `cov1` $= 0.9$, instead of -0.9. Repeat part (a) on this new data. If everything is working correctly, the classifier should have two *sharply* curved decision boundaries defining two blue regions and one red region.

(d) Repeat part (c), but in the training set, put 1,000 points in class 0 and 5,000 in class 1; and in the test set, 10,000 points in class 0 and 50,000 in class 1. If everything is working correctly, the classifier should have two *slightly* curved decision boundaries defining one blue region and two red regions.

(e) Using the theory developed in Question 3 as a guide, write a Python function `myGDA(Xtrain,Ttrain,Xtest,Ttest)` that performs Gaussian Discriminant Analysis for two classes. However, do *not* assume the two classes share the same covariance matrix. Instead, adapt the equations from Question 3 as necessary. *Write down and hand in any adapted equations that you use,* but do not prove them.

`myGDA` should fit a Gaussian Bayes classifier to the training data, and compute and return its accuracy on the test data. The function should work on any 2-dimensional data for any two-class problem, even if it was not produced by `gen_data`. *i.e.*, it should work on real data as well as synthetic data.

The input to `myGDA` has the same form as the data returned by `gen_data`. In particular, each row of `Xtrain` is an input vector, and each element of `Ttrain` is the corresponding class label (0 or 1), where the data in these matrices is not in any particular order. They form the training data. Likewise, `Xtest` and `Ttest` form the test data.

Use `myGDA` to train and test a binary classifier using the data from Question 4(a). Call the test accuracy `accuracy4e`. It should be the same as the accuracy from Question 4(a) (call it `accuracy4a`). Print out `accuracy4e`, `accuracy4a` and their difference. The difference should be very close to 0 (*e.g.*, less than 10^{-4} in magnitude).

You should not use any built-in methods to compute probabilities, means, covariance matrices or any other statistical quantities. You should also not use any functions in `scipy` or `sklearn`. You should only use `numpy` functions, but no statistical functions, such as `numpy.mean`, `numpy.var`, `numpy.cov` or `numpy.std`, nor any functions in `numpy.random` or `numpy.linalg`, except as specified below. The idea is to do as much as possible from scratch by using basic `numpy` methods such as `sum`, `matmul`, `exp`, `max` and `argmax`. You may, however, use `inv` and `det` in `numpy.linalg`.

The entire question can be done in at most 40 lines of code, with no loops.

5. (? points total) *Naive Bayes*

In this question, you will use full and naive Gaussian Bayes to classify images of handwritten digits. There are ten different digits (0 to 9), so you will be using multi-class classification.

To start, download and uncompress (if necessary) the MNIST data file from the course web page. The file, called `mnist.pickle.zip`, contains training and test data. Next, start the Python interpreter and import the `pickle` module. You can then read the file `mnist.pickle` with the following command (`'rb'` opens the file for reading in binary):

```
with open('mnist.pickle','rb') as f:
        Xtrain,Ttrain,Xtest,Ttest = pickle.load(f)
```

The variables `Xtrain` and `Ttrain` contain training data, while `Xtest` and `Test` contain test data. Use this data for training and testing in this question and in the rest of this assignment.

`Xtrain` is a Numpy array with 60,000 rows and 784 columns. Each row represents a hand-written digit. Although each digit is stored as a row vector with 784 components, it actually represents an array of pixels with 28 rows and 28 columns ($784 = 28 \times 28$). Each pixel is stored as a floating-point number, but has an integer value between 0 and 255 (i.e., the values representable in a single byte). The variable `Ttrain` is a vector of 60,000 image labels, where a label is an integer betwen 0 and 9. For example, if row `n` of `Xtrain` is an image of the digit 7, then `Ttrain[n]` = 7. Likewise for `Xtest` and `Ttest`, which represent 10,000 test images.

To view a digit, you must first convert it to a 28×28 array using the function `numpy.reshape`. To display a 2-dimensional array as an image, you can use the function `imshow` in `matplotlib.pyplot`. To see an image in black-and-white, add the keyword argument `cmap='Greys'` to `imshow`. To remove the smoothing and see the 784 pixels clearly, add the keyword argument `interpolation='nearest'`. Try displaying a few digits as images. (Figure 3 shows an example.) For comparison, try printing them as vectors. (Do not hand this in.)

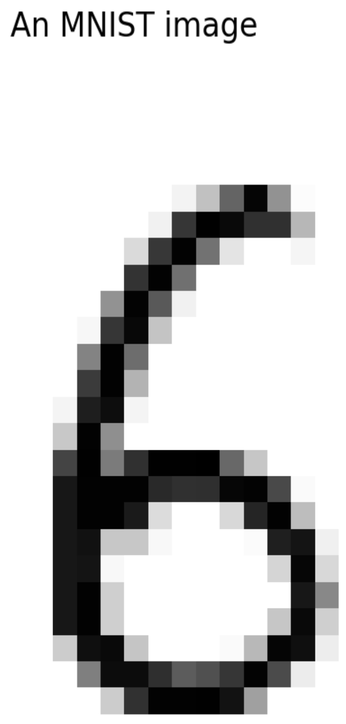
What to do: Write Python programs to carry out the following tasks:

(a) (? points) Choose 25 MNIST images at random (without replacement) and display them in a single figure, arranged in a 5×5 grid. Turn off the axes in each image using the function `matplotlib.pyplot.axis`. The images should be in black-and-white and should not use any smoothing. Title the figure, "Question 5(a): 25 random MNIST images." You may use one loop for this question.

(b) (? points) As in Question 4, use the Python class `QuadraticDiscriminantAnalysis` in `sklearn.discriminant_analysis` to train a full Gaussian Bayes classifier on the MNIST training data. Using the methods and attributes of this class, compute and print out the training and test accuracies of the classifier on the MNIST data. In addition, measure and print out the amount of time required to fit the model to the training data. (You will probably get a run-time warning message. You can ignore it, but include it in the output that you hand in. No points will be deducted.) You can easily do this in 12 lines of code without loops. You should find that the training and test accuracies are both about 17%.

(c) (? points) Repeat part (b) using the Python class `GaussianNB` in `sklearn.naive_bayes` to train a Gaussian naive Bayes classifier on the MNIST training data. You can easily do this in 2 lines of code without loops by reusing code from part (b). You should find that the training and test accuracies are both about 55%. You should also find that naive Bayes is about 10 times faster than full Bayes.

Figure 3:



(d) Modify the MNIST training data by using the following code to add a small amount of Gaussian noise to the data:

```
sigma = 0.1
noise = sigma*np.random.normal(size=np.shape(Xtrain))
Xtrain = Xtrain + noise
```

Now repeat part (a), retitling the figure appropriately. You should find that the images look slightly noisy. In the rest of this question, we will use only the noisy data.

(e) Repeat parts (b) and (c) using the noisy data. You should find that the classifiers trained on the noisy data are much more accurate, and that full Gaussian Bayes now has a better test accuarcy than Gaussian naive Bayes. You can do this in 4 lines of code. *Briefly explain why you think adding noise improves the accuracy of the classifiers. Could this be related to any warning messsages you received?*

(f) Repeat part (e) using only the first 6000 elements of the noisy training data (*i.e.*, reduce the training data by %90). Continue to use all of the test data. You can easily do this in at most 7 lines of code. You should find that the test accuracy for naive Bayes is about the same as in part (e), while the test accuracy for full Bayes has plummeted. *Provide an interpretation of all the training and test accuracies in this part and in part (e).*

(g) For each digit class, Gaussian naive Bayes estimates a 784-dimensional mean vector, μ. Display the ten mean vectors as 28×28 images in a 3×4 grid. Each image should look like a fuzzy digit on a white background. Title the figure, "Question 5(g): means for each digit class." You may use one loop for this question, though none are needed if you reuse code from earlier questions.

Using these images, give a simple and brief interpretation of what Gausian naive Bayes does. In the mean images, the fuzzy digits appear on a white background, whereas the training digits have a noisy background, as seen in part (d). *What happened to the background noise?*

(h) Write a Python function `myGNB(Xtrain,Ttrain,Xtest,Ttest)` that performs Gaussian naive Bayes for multi-class classification. The function should work for any number of classes and data of any dimensionality (not just MNIST data). The input has the same form as the input to `myGDA` in Question 4(e), except that `Ttrain` and `Ttest` can contain any integer (class label) from 0 to K-1, where K is the number of classes. The function should fit a Gaussian naive Bayes classifier to the training data and return both the training and test accuracies.

Apply `myGNB` to the reduced MNIST data of part (f). Print the training and test accuracies. They should be identical to the accuracies for naive Bayes in part (f). Print out the difference in the two training accuracies and the difference in the two test accuracies. The difference in test accuracies should be zero (or very close to it). The difference in training accuracies may be non-zero, but should be below 0.001 (This is due to numerical error, which may cause a run-time warning, which you can ignore.)

Your code for `myGNB` should follow the same restrictions as that for `myGDA` in Question 4(e). In addition, you should *not* use any built-in functions for matrix multiplication, matrix inverse or determinants (or anything equivalent to them, such as `numpy.dot`). They are not needed for naive Bayes, and something much simpler and computationally more-efficient should be used.

The entire question can be done in at most 45 lines of code. You may use *at most one loop.*

? points total

University of Toronto Mississauga
CSC 411 - Machine Learning and Data Mining

Cover sheet for Assignment 2

Complete this page and hand it in with your assignment.

Name: ______________________________
(Underline your last name)

Student number: ______________________________

I declare that the solutions to Assignment 2 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: ______________________________