Assignment 2: Questions and Solutions

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 programs, 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 labeled 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 solutions
to Assignment 2.

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.
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([x1, x2,..., xn]) returns the list [f(x1), f(x2),..., f(xn)]. 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. (10 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 \texttt{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 \( \text{mu0} \) and covariance \( \text{cov0} \). The cluster for class 1 has \( N1 \) points, mean \( \text{mu1} \) and covariance \( \text{cov1} \). Note that \( \text{mu0} \) and \( \text{mu1} \) and all the data points are 2-dimensional vectors, while \( \text{cov0} \) and \( \text{cov1} \) 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

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

where \( \mu \) is the mean (cluster centre), \( \Sigma \) is the covariance matrix, and \( k \) is the dimensionality of the data (2 in our case). To generate data for a cluster, use the function \texttt{multivariate.normal} in \texttt{numpy.random}. Use the function \texttt{shuffle} in \texttt{sklearn.utils} to distribute the data randomly in the arrays. You should be able to write \texttt{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 \( \sigma_1^2 = \text{var}(x_1) \), \( \sigma_2^2 = \text{var}(x_2) \) and \( \sigma_{12} = \text{cov}(x_1, x_2) \). For this problem, \( \text{var}(x_1) = \text{var}(x_2) = 1 \) for both classes, and \( \text{cov}(x_1, x_2) \) is \( \text{cov0} \) for class 0, and \( \text{cov1} \) for class 1.

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

(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 \texttt{scatter} in \texttt{numpy.pyplot}. Specify a relatively small dot size by using the named argument \texttt{s=2}. Use the functions \texttt{xlim} and \texttt{ylim} to extend the \( x \) and \( y \) axes from -3 to 6. Adjust the
2. (25 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) (5 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) (5 points) Compute the accuracy of your logistic-regression classifier on the train-
ing 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, re-
spectively. 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) (5 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 this in at most 7 lines of code
with no loops.

(e) (5 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.

ANSWER: Suppose we wish to draw the contour specified by $P(C = 1|x) = p,$
for some constant, $p$. Then $p = 1/(1 + \exp(-z))$, where $z = w^T x + w_0$. Solving for
$z$ gives, $z = \log(1/p - 1)$. Thus, the contour is given by the equation, $w^T x + w_0 =
\log(1/p - 1)$, or equivalently, $w^T x + w'_0 = 0$, where $w'_0 = w_0 - \log(1/p - 1)$.
Thus, to draw the contour, we can simply call plotDB($w, w'_0$), using the function
plotDB from part (d).
(f) (0 points) Use \texttt{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) (5 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. (25 points total) \textit{Generative Models: Theory}

(a) (10 points) Prove the second equation on slide 18 of lecture 9:

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

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.)

\textbf{ANSWER:} The loss function is the negative log likelihood and is given by

$$
\ell = -\log \prod_{n=1}^{N} p(x^{(n)}; t^{(n)}) \quad \text{from slide 17 of Lecture 9}
$$

$$
= -\log \prod_{n=1}^{N} p(x^{(n)}|t^{(n)}) p(t^{(n)})
$$

$$
= -\sum_{n=1}^{N} \log[p(x^{(n)}|t^{(n)}) p(t^{(n)})]
$$

$$
= -\sum_{n=1}^{N} [\log p(x^{(n)}|t^{(n)}) + \log p(t^{(n)})]
$$

$$
= -\sum_{n=1}^{N} [\log p(x^{(n)}|t^{(n)}) + \log[\phi^{t^{(n)}}(1 - \phi)^{1-t^{(n)}}]] \quad \text{by the top equation on slide 18 of Lecture 9}
$$

$$
= -\sum_{n=1}^{N} [\log p(x^{(n)}|t^{(n)}) + t^{(n)} \log \phi + (1 - t^{(n)}) \log(1 - \phi)]
$$

In addition, $p(x^{(n)}|t^{(n)})$ depends on $\mu_0$, $\mu_1$ and $\Sigma$, but not on $\phi$ (slide 17, lecture 9). Thus,

$$
\frac{\partial \ell}{\partial \phi} = -\sum_{n=1}^{N} \frac{\partial [t^{(n)} \log \phi + (1 - t^{(n)}) \log(1 - \phi)]}{\partial \phi}
$$
\[- \sum_{n=1}^{N} \left[ e(r(n)) \frac{\partial \log \phi}{\partial \phi} + (1 - t^{(n)}) \frac{\partial \log (1 - \phi)}{\partial \phi} \right] \]

\[- \sum_{n=1}^{N} \left[ \frac{e(r(n))}{\phi} - \frac{(1 - t^{(n)})}{(1 - \phi)} \right] \]

\[- \sum_{n=1}^{N} \left[ \frac{e(r(n)) (1 - \phi) - (1 - t^{(n)}) \phi}{\phi (1 - \phi)} \right] \]

\[- \sum_{n=1}^{N} \frac{e(r(n)) - \phi i}{\phi (1 - \phi)} \]

\[= \frac{\left| N \phi - \sum_{n=1}^{N} t^{(n)} \right|}{\phi (1 - \phi)} \]

Thus,

\[\frac{\partial \ell}{\partial \phi} = 0 \text{ iff } N \phi - \sum_{n=1}^{N} e(r(n)) = 0 \]

\[\text{iff } \phi = \sum_{n=1}^{N} t^{(n)}/N \]

\[\text{iff } \phi = \sum_{n=1}^{N} I[e(r(n)) = 1]/N \]

since \(t = I[t = 1]\), since \(t \in \{0, 1\}\). Thus, the optimal value of \(\phi\) is \(\sum_{n=1}^{N} I[e(r(n)) = 1]/N\).

(b) (10 points) Consider the following version of the last equation on slide 18 of Lecture 9:

\[\Sigma = \frac{1}{N} \sum_{n=1}^{N} (x^{(n)} - \mu_{t^{(n)}}, x^{(n)} - \mu_{t^{(n)}})^T \]

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

\[\Sigma = \frac{1}{N} \left[ Y_0^T Y_0 + Y_1^T Y_1 \right]/N \]

where \(Y_i = X_i - \bar{1} \mu_i^T\), where \(\mu_i\) is treated as a column vector and \(\bar{1}\) is a column vector of 1’s.
ANSWER: Given a matrix, $A$, let $[A]_n$ denote the $n^{th}$ row vector of $A$, and let $[A]_{ij}$ denote the $(i,j)^{th}$ element of $A$. Note that if $v$ is a row vector, then $v^T v$ is a matrix where $(v^T v)_{ij} = v_i v_j$. Letting $v = [A]_n$, we get $([A]^T[A]_n)_{ij} = [A]_n[A]_{nj}$.

We can therefore prove the following result:

$$[A^T A]_{ij} = \sum_n [A^T]_m[A]_{nj} \quad \text{by the definition of matrix multiplication}$$

$$= \sum_n [A]_m[A]_{nj} \quad \text{by the definition of matrix transpose}$$

$$= \sum_n ([A]^T[A]_n)_{ij} \quad \text{by the observation above}$$

$$= \sum_n [A]^T[A]_n$$

Thus, $A^T A = \sum_n [A]^T[A]_n$ since all their components are equal. In addition, note that if $v$ is a row vector, then $\hat{v}^T v$ is a matrix in which every row is equal to $v$. Thus, $[\hat{v}^T v]_n = v$ for all $n$. Keeping these points in mind,

$$N\Sigma = \sum_{n=1}^N (x^{(n)} - \mu_{i^{(n)}})^T (x^{(n)} - \mu_{i^{(n)}}) \quad \text{by definition}$$

$$= \sum_{n:X^{(n)} = 0} (x^{(n)} - \mu_0)^T (x^{(n)} - \mu_0) + \sum_{n:X^{(n)} = 1} (x^{(n)} - \mu_1)^T (x^{(n)} - \mu_1)$$

$$= \sum_{x \in X_0} (x - \mu_0)^T (x - \mu_0) + \sum_{x \in X_1} (x - \mu_1)^T (x - \mu_1)$$

$$= \sum_{n=1}^{N_0} ([X_0]_n - \mu_0)^T ([X_0]_n - \mu_0) + \sum_{n=1}^{N_1} ([X_1]_n - \mu_1)^T ([X_1]_n - \mu_1)$$

$$= \sum_{n=1}^{N_0} ([X_0]_n - \hat{\mu}_0)_n^T ([X_0]_n - \hat{\mu}_0)_n + \sum_{n=1}^{N_1} ([X_1]_n - \hat{\mu}_1)_n^T ([X_1]_n - \hat{\mu}_1)_n$$

since $\mu_i = [\hat{\mu}_i]_n$, as noted above

$$= \sum_{n=1}^{N_0} [X_0 - \hat{\mu}_0]_n^T [X_0 - \hat{\mu}_0]_n + \sum_{n=1}^{N_1} [X_1 - \hat{\mu}_1]_n^T [X_1 - \hat{\mu}_1]_n$$

$$= \sum_{n=1}^{N_0} [Y_0]_n^T [Y_0]_n + \sum_{n=1}^{N_1} [Y_1]_n^T [Y_1]_n \quad \text{by the definition of } Y_i$$

$$= Y_0^T Y_0 + Y_1^T Y_1 \quad \text{by the result above}$$

Thus $\Sigma = (Y_0^T Y_0 + Y_1^T Y_1)/N$.

(c) (5 points) Convert the third and fourth equations on slide 18 to matrix form. That is, derive matrix equations for $\mu_0$ and $\mu_1$. 

8
ANSWER: The third equation on slide 18 has the form $\mu_0 = s/N_0$, where $N_0$ is the number of points in class 0, and $s$ is the sum of all the input vectors in class 0. $s$ is thus the sum of all the rows of $X_0$, the data matrix for class 0, defined in part (b). That is, $s = \sum_n [X_0]_n$. Thus,

$$
[s]_i = \sum_n [X_0]_{ni} \\
= \sum_n [\bar{1}^T]_n [X_0]_{ni} \quad \text{since} \quad [\bar{1}^T]_n = 1 \\
= [\bar{1}^T X_0]_i \quad \text{by the definition of vector-matrix multiplication}
$$

Thus $s = \bar{1}^T X_0$, since all their components are equal. Thus, since $\mu_0 = s/N_0$, we get the matrix equation $\mu_0 = \bar{1}^T X_0/N_0$.

Similarly, we get the matrix equation $\mu_1 = \bar{1}^T X_1/N_1$, where $X_1$ is the data matrix for class 1, and $N_1$ is the number of rows in $X_1$.

4. (30 points total) Generative Models: Practice

In this question, you will build 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.\footnote{\textit{dfContour} plots the decision boundary as a black line and plots several other contours of $P(C = 1|x)$ for a classifier. The color 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 $C = 0$.}

(a) (3 points) Using the Python class QuadraticDiscriminantAnalysis in sklearn.discriminant_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 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) (6 points) 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.

ANSWER: Consider the dashed diagonal line in Figure 3. If we slice the red and blue clusters along this line, their probability densities appear as 1-dimensional Gaussians, as shown in Figure 4. The two curved decision boundaries in Figure 3 touch the dashed line at points A and B in Figure 4. At these two points, the two
Figure 2:

Question 4(a): Decision boundary and contours
Gaussians have equal probability. Note how the tall narrow Gaussian of the blue cluster divides the wide Gaussian of the red cluster into two regions, one to the left of point A and one to the right of point B. These correspond to the two red regions in Figure 3. Here, the red cluster has greater probability than the blue cluster, as seen in Figure 4. The region between points A and B corresponds to the blue region in Figure 3. Here, the blue cluster has greater probability, as also seen in Figure 4.

(c) (3 points) Generate new training and test data that is the same as that of Question 2 except that \( \text{cov1} = 0.9 \), 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.
Figure 4:

(d) (3 points) 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) (15 points) 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 `gendata`. *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 `gendata`. 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.

ANSWER: Let \( N_i \) be the number of rows in matrix \( Y_i \). Then the two covariance matrices are given by:

\[
\Sigma_0 = Y_0^T Y_0 / N_0 \quad \Sigma_1 = Y_1^T Y_1 / N_1
\]

5. (50 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):

```python
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 between 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 Test, which represent 10,000 test images.

To view a digit, you must first convert it to a \(28 \times 28\) 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 5 shows an example.) For comparison, try printing them as vectors. (Do not hand this in.)
Figure 5:
An MNIST image
What to do: Write Python programs to carry out the following tasks:

(a) (5 points) Choose 25 MNIST images at random (without replacement) and display them in a single figure, arranged in a $5 \times 5$ 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) (3 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) (3 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.

(d) (1 point) Modify the MNIST training data by using the following code to add a small amount of Gaussian noise to the data:

```python
sigma = 0.1
noise = sigma*np.random.normal(size=Xtrain.shape)
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) (8 points total) 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 accuracy than Gaussian naive Bayes. You can do this in 4 lines of code. (3 points)

*Briefly explain why you think adding noise improves the accuracy of the classifiers. Could this be related to any warning messages you received?* (5 points)

**Answer:** Without the noise, the background in all the figures has an almost constant value of 0. The variance of these background pixels is therefore almost 0. Using `numpy.cov` to compute the covariance matrix of the training data confirms this: many pixels have zero variance, and many more have a variance very close to zero (e.g., $10^{-5}$ or less). This causes many numerical problems. In particular,
it is impossible to compute the probability density of a background pixel with zero variance, since the formula for Gaussian probability density is

\[ P(x) = \frac{e^{-(x-\mu)^2/2\sigma^2}}{\sqrt{2\pi\sigma}} \]

which is undefined at \( \sigma = 0 \). This means we cannot evaluate the probability of an image, since we cannot compute the probability of all its pixels. Presumably the function \texttt{GaussianNB} in \texttt{sklearn} has some mechanism for handling this (possibly through regularization), since no errors occurred, but in any case, the learning seems to be compromised. It also explains the warning message.

In addition, because most of the remaining background pixels have a variance near zero, their probabilities tend to be extremely large, because we divide by \( \sigma \) in the above probability formula. This means that the probability of an image is largely determined by its background pixels, which contain little information about the class of the image. Again, the learning is compromised. Hence the low accuracies.

In contrast, when noise is added, the background is not constant. The noise gives each background pixel a mean of zero and a standard deviation of 0.1. In fact, it guarantees that all pixels have a significant, non-zero variance. It is therefore possible to compute the probability density of all the pixels in an image, and thus of the image itself, without numerical error or compromise. Also, the probability of an image is no longer dominated by its background pixels. Learning is therefore not compromised. Hence the much higher accuracies.

(f) (7 points total) 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. (3 points)

Provide an interpretation of all the training and test accuracies in this part and in part (e). (4 points)

ANSWER: In part (e), the full Bayesian model has many more parameters and can form a much more complex decision boundary than naive Bayes. Hence, its training and test accuracies are both significantly higher than for naive Bayes. In this case, the relatively simplistic naive Bayes seems to be underfitting the data. In part (f), there is much less training data, and the full Bayesian model is grossly overfitting the data, having a perfect training accuracy of 100% but a test accuracy of only 17%. In contrast, naive Bayes, which has far fewer parameters, is able to handle the smaller training set without overfitting and has virtually the same training and test accuracies as in part (e).

(g) (7 points total) For each digit class, Gaussian naive Bayes estimates a 784-dimensional mean vector, \( \mu \). Display the ten mean vectors as 28 \times 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. (3 points)

*Using these images, give a simple and brief interpretation of what Gaussian 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?* (4 points)

**ANSWER:** Since the mean images are fuzzy digits, this suggests that naive Bayes is a form of template matching: the fuzzy images act as templates, one for each digit class; and for each image, \( x \), naive Bayes checks the image against each template to see which template matches the image best.

In the background pixels of the mean images, we are seeing the mean of the noise, not the noise itself. Since the mean of the Gaussian noise is zero, the background appears white in the mean images.

(h) (16 points) 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.*

140 points total
import numpy as np
import numpy.random as rnd
import numpy.linalg as la
import matplotlib.pyplot as plt
import sklearn.linear_model as lin
import sklearn.utils as utils
import pickle
import time
import sklearn.discriminant_analysis as da
import sklearn.naive_bayes as nb
import bonnerlib2 as bonner

##### Question 1 #####

# Question 1(a)
# generate cluster data

def gen_data(mu0,mu1,cov0,cov1,N0,N1):
    Sigma0 = [[1,cov0],[cov0,1]]
    Sigma1 = [[1,cov1],[cov1,1]]
    data0 = rnd.multivariate_normal(mu0,Sigma0,N0)  # data mat:
    data1 = rnd.multivariate_normal(mu1,Sigma1,N1)  # data mat:
    # combine the two data matrices into one
    X = np.concatenate([data0,data1])
    # construct a vector of labels
    t0 = np.zeros([N0],dtype=np.int32)  # labels for cluster 0
    t1 = np.ones([N1],dtype=np.int32)  # labels for cluster 1
    # combine the two vectors into one
    t = np.concatenate([t0,t1])
    return utils.shuffle(X,t)  # shuffle and return the data

# Question 1(b)
# generate cluster data
N0 = 10000
N1 = 5000
mu0 = [1.0,1.0]
mu1 = [2.0,2.0]
cov0 = 0
cov1 = -0.9
X,T = gen_data(mu0,mu1,cov0,cov1,N0,N1)
# Question 1(c)
# display cluster data
xmin = -3
xmax = 6

def plot_data(X,T):
    plt.figure()
    colors = np.array(['r','b'])  # red for class 0, blue for
    plt.scatter(X[:,0], X[:,1], color=colors[T], s=2)
    plt.xlim(xmin, xmax)
    plt.ylim(xmin, xmax)

plot_data(X,T)
plt.suptitle('Question 1(c): sample cluster data')


######## Question 2 ########

# Question 2(a)
# generate training data
N0 = 1000
N1 = 500
Xtrain,Ttrain = gen_data(mu0,mu1,cov0,cov1,N0,N1)

# Question 2(b)
# fit a logistic-regression classifier to the data
print('
')
print('Question 2(b).')
print('-------------')

clf = lin.LogisticRegression()
clf.fit(Xtrain,Ttrain)  # learn a logistic-regression classifier
w = clf.coef_[0]  # weight vector
w0 = clf.intercept_[0]  # bias term
print('w0 =',w0)
print('w =',w)

# Question 2(c)
print('
')
print('Question 2(c).')
print('-------------')
# Compute accuracy in two different ways.
# (1) using the score method
accuracy1 = clf.score(Xtrain, Ttrain)

# (2) from w and w0
z = np.matmul(Xtrain, w) + w0  # values of the linear function:
Predictions = (z>0)  # predictions for each training point (True/False)
Truth = Ttrain

correct = (Predictions == Truth)  # correct predictions
accuracy2 = np.mean(correct)

# print results
print 'accuracy1 =', accuracy1
print 'accuracy2 =', accuracy2
print 'accuracy1 - accuracy2 =', accuracy1-accuracy2

# Question 2(d)
# plot the training data and the decision boundary

def plotDB(w, w0, colour):
    y1 = -(w0+w[0]*xmin)/w[1]  # the left-most point in the fit
    y2 = -(w0+w[0]*xmax)/w[1]  # the right-most point in the fit
    # draw a line between the left- and right-most points.
    plt.plot([xmin, xmax], [y1, y2], colour)

plot_data(Xtrain, Ttrain)
plotDB(w, w0, 'k')
plt.suptitle('Question 2(d): training data and decision boundary

# Question 2(e)
# plot the training data and several contours

def plot_contour(w, w0, p, colour):
    t = np.log(p/(1-p))
    plotDB(w, w0-t, colour)

plot_data(Xtrain, Ttrain)
plot_contour(w, w0, 0.6, 'b')
plot_contour(w, w0, 0.5, 'k')
plot_contour(w, w0, 0.05, 'r')
plt.suptitle('Question 2(e): three contours

# Question 2(f)
# generate test data

N0 = 10000
N1 = 5000
Xtest, Ttest = gen_data(mu0, mu1, cov0, cov1, N0, N1)

# Question 2(g)
# precision and recall
print('
')
print('Question 2(g).')
print('--------------

# compute the precision and recall for the contour P(C=1|x) = p

def precision_recall(w, w0, p):
    z = np.matmul(Xtest, w) + w0  # compute z values for each test point
    t = np.log(p/(1-p))          # threshold on z values for the contour
    # classify each test point as positive, predicted positive,
    # positives
    Pos = Ttest                
    PP = (z>=t)                
    TP = Pos & PP             
    # true positive predictions
    numPP = np.sum(PP)         
    numTP = np.sum(TP)         
    # number of predicted positives
    # number of true positives

    Precision = numTP/np.float(numPP)
    Recall = numTP/np.float(np.sum(Pos))
    print('
' + For P(C=1|x) = ', p
    print('  Precision = ', Precision
    print('  Recall = ', Recall

precision_recall(w, w0, 0.6)
precision_recall(w, w0, 0.5)
precision_recall(w, w0, 0.05)


######  Question 4  ######

# Question 4(a)
print('
')
print('Question 4(a).')
print('--------------

def myfit4a():
    clf = da.QuadraticDiscriminantAnalysis()
    clf.fit(Xtrain, Ttrain)
    acc = clf.score(Xtest, Ttest)
    print 'Test accuracy = ', acc

4
plot_data(Xtrain,Ttrain)
    bonner.dfContour(clf)
    plt.title('Decision boundary and contours')
    return acc

accuracy4a = myfit4a()
plt.suptitle('Question 4(a):')

Data4a = [Xtrain,Ttrain,Xtest,Ttest]  # save the data for Ques:

# Question 4(c)
print('
')
print('Question 4(c):')
print('------------')
# generate new data
N0 = 1000
N1 = 500
cov0 = 0
cov1 = 0.9
Xtrain,Ttrain = gen_data(mu0,mul,cov0,cov1,N0,N1)
N0 = 10000
N1 = 5000
Xtest,Ttest = gen_data(mu0,mul,cov0,cov1,N0,N1)

accuracy4c = myfit4a()
plt.suptitle('Question 4(c):')

# Question 4(d)
print('
')
print('Question 4(d):')
print('------------')
# generate new data
N0 = 1000
N1 = 5000
Xtrain,Ttrain = gen_data(mu0,mul,cov0,cov1,N0,N1)
N0 = 10000
N1 = 50000
Xtest,Ttest = gen_data(mu0,mul,cov0,cov1,N0,N1)

accuracy4d = myfit4a()
plt.suptitle('Question 4(d):')
# Question 4(e)
print(' 
')
print('Question 4(e).')
print('--------------

# return the probability density of each row-vector in X
# for a multivariate normal distribution with mean mu and covari.
def mvn_pdf(X,mu,Sigma):
    SigmaInv = la.inv(Sigma)  # matrix inverse of Sigma
    detSigma = la.det(Sigma)  # determinant of Sigma
    N,M = np.shape(X)
    mu = np.reshape(mu,[1,M])
    Y = X-mu
    Z = np.matmul(Y,SigmaInv)
    Q = np.sum(Z*Y,axis=1)  # quadratic exponent of Gaussian
    p = np.exp(-Q/2)/(2*np.pi*np.sqrt(detSigma))  # vector of 1
    return p

# fit a multivariate normal to the data in X
def fit_mvn(X):
    N,M = np.shape(X)
    mu = np.sum(X,axis=0)/N
    mu = np.reshape(mu,[1,M])
    X = X-mu
    Sigma = np.matmul(X.T,X)/N
    return mu,Sigma

# Train a GDA classifier and return its test accuracy
def myGDA(Xtrain,Ttrain,Xtest,Ttest):

    # construct a data matrix for each class
    X1 = Xtrain[Ttrain==1]  # data matrix for class 0
    X0 = Xtrain[Ttrain==0]  # data matrix for class 1

    # fit a multivariate Gaussian to each data set
    mu0,Sigma0 = fit_mvn(X0)
    mu1,Sigma1 = fit_mvn(X1)

    # estimate prior probabilities
    N0 = np.shape(X0)[0]  # number of training points in class 0
    N1 = np.shape(X1)[0]  # number of training points in class 1
    prior0 = N0/float(N0+N1)  # prior for class 0
    prior1 = 1-prior0  # prior for class 1
# Use Bayes rule to predict the class of each row vector in:
P0 = mvn_pdf(Xtest, mu0, Sigma0) * prior0  # posterior probability
P1 = mvn_pdf(Xtest, mu1, Sigma1) * prior1  # posterior probability
predictions = np.argmax([P0, P1], axis=0)

# compute accuracy of the predictions
Ntest = np.shape(Xtest)[0]
accuracy = np.sum(predictions==Ttest)/float(Ntest)
return accuracy

Xtrain, Ttrain, Xtest, Ttest = Data4a  # use the data from Question 4
accuracy4e = myGDA(Xtrain, Ttrain, Xtest, Ttest)

print 'accuracy4e =', accuracy4e
print 'accuracy4a =', accuracy4a
print 'accuracy4e - accuracy4a =', accuracy4e - accuracy4a

######## Question 5 #######

# read MNIST data from a file
with open('mnist.pickle', 'r') as f:
    mnist = pickle.load(f)
Xtrain, Ttrain, Xtest, Ttest = mnist

# Question 5(a)
# display the rows of data as images arranged in a LxW grid
def displayImages(data, L, W):
    plt.figure()
    N, M = np.shape(data)
    # number of rows and columns in each image
    m = int(np.sqrt(M))
data = np.reshape(data, [N, m, m])
for i in range(0, N):
    # display the ith image
    plt.subplot(L, W, i+1)
    plt.axis('off')
    plt.imshow(data[i], cmap='Greys', interpolation='nearest')
```python
displayImages(sample, 5, 5)  # display the 25 elements
plt.suptitle('Question 5(a): 25 random MNIST images')

# Question 5(b)
print('\\n')
print('Question 5(b).')
print('-------------------')

def myFit5(clf, Xtrain, Ttrain, Xtest, Ttest):
    t1 = time.time()
    clf.fit(Xtrain, Ttrain)
    t2 = time.time()
    print('fitting time =', t2-t1)
    accuracyTrain = clf.score(Xtrain, Ttrain)
    accuracyTest = clf.score(Xtest, Ttest)
    print('Training accuracy =', accuracyTrain)
    print('Test accuracy =', accuracyTest)
    return accuracyTrain, accuracyTest

# full Gaussian Bayes
clf = da.QuadraticDiscriminantAnalysis()
myFit5(clf, Xtrain, Ttrain, Xtest, Ttest)

# Question 5(c)
print('\\n')
print('Question 5(c).')
print('-------------------')
# Gaussian naive Bayes
clf = nb.GaussianNB()
myFit5(clf, Xtrain, Ttrain, Xtest, Ttest)

# Question 5(d)
# add noise to the training data
sigma = 0.1
noise = sigma*np.random.normal(size=np.shape(Xtrain))
Xtrain = Xtrain + noise

displayImages(Xtrain[::25], 5, 5)
plt.suptitle('Question 5(d): 25 noisy MNIST images')
```
# Question 5(e)
print('\n')
print('Question 5(e).')
print('----------')
# full Gaussian Bayes
print "Full Gaussian Bayes classifier:'
clf = da.QuadraticDiscriminantAnalysis()
myFit5(clf,Xtrain,Ttrain,Xtest,Ttest)

print '\nGaussian naive-Bayes classifier:'
# Gaussian naive Bayes
clf = nb.GaussianNB()
myFit5(clf,Xtrain,Ttrain,Xtest,Ttest)

# Question 5(f)
print('\n')
print('Question 5(f).')
print('----------')

N = 6000
Xtrain2 = Xtrain[:N]
Ttrain2 = Ttrain[:N]

# full Gaussian Bayes
print "Full Gaussian Bayes classifier:'
clf = da.QuadraticDiscriminantAnalysis()
myFit5(clf,Xtrain2,Ttrain2,Xtest,Ttest)

# Gaussian naive Bayes
print '\nGaussian naive-Bayes classifier:'
clf = nb.GaussianNB()
accuracyTrain5f,accuracyTest5f = myFit5(clf,Xtrain2,Ttrain2,Xtest5f)

# Question 5(g)
# display class means
means = clf.theta
displayImages(means,3,4)
plt.suptitle('Question 5(g): means for each digit class')

# Question 5(h)
print('
')
print('Question 5(h).')
print('------------

# return the probability density of each row-vector in X
# for an axis-aligned Gaussian with mean vector mu and variance

def aag_prob(X, mu, var):
    N, M = np.shape(X)
    mu = np.reshape(mu, [1, M])
    var = np.reshape(var, [1, M])
    Q = (X-mu)**2/(2*var)   # NxM matrix of quadratic Gaussian
    Z = np.sqrt(2*np.pi*var) # 1xM matrix of normalization constant
    prob = np.exp(-Q)/Z     # NxM matrix of probabilities
    return np.prod(prob, axis=1) # N-vector of probabilities

# fit an axis-aligned Gaussian to the data in X

def fit_aag(X):
    N, M = np.shape(X)
    mu = np.sum(X, axis=0)/N
    mu = np.reshape(mu, [1, M])
    var = np.sum((X-mu)**2, axis=0)/N
    return mu, var

# Train a Gaussian naive-Bayes classifier and return its training predictions

def myGNB(Xtrain, Ttrain, Xtest, Ttest):
    Ntrain, M = np.shape(Xtrain)
    Ntest, M = np.shape(Xtest)
    K = np.max(Ttrain) + 1  # number of classes
    Mu = np.zeros([K, M])
    Var = np.ones([K, M])
    Prior = np.ones(K)
    # reserve space for class conditional probabilities
    PcondTrain = np.zeros([K, Ntrain])
    PcondTest = np.zeros([K, Ntest])
    # fit a Gaussian to each class
    for k in range(K):
        X = Xtrain[Ttrain==k]  # data matrix for class K
        Mu[k], Var[k] = fit_aag(X)  # mean and variance for class
        N = np.shape(X)[0]       # number of training points for class
        Prior[k] = N/float(Ntrain)  # prior probability for class
        PcondTrain[k] = aag_prob(Xtrain, Mu[k], Var[k])  # class
        PcondTest[k] = aag_prob(Xtest, Mu[k], Var[k])    # class
    # apply Bayes rule and make predictions
    Prior = np.reshape(Prior, [K, 1])
    PostTrain = PcondTrain*Prior  # NxK matrix of posterior prob


PostTest = PcondTest*Prior
predictionsTrain = np.argmax(PostTrain,axis=0)
predictionsTest = np.argmax(PostTest,axis=0)
# compute accuracy of the predictions
accuracyTrain = np.sum(predictionsTrain==Ttrain)/float(Ntrain)
accuracyTest = np.sum(predictionsTest==Ttest)/float(Ntest)
return accuracyTrain,accuracyTest

accuracyTrain,accuracyTest = myGNB(Xtrain2,Ttrain2,Xtest,Ttest)

print 'Training accuracy =', accuracyTrain
print 'Test accuracy =', accuracyTest
print "
print 'Difference in training accuracies =', accuracyTrain - accuracyTest
print 'Difference in test accuracies =', accuracyTest - accuracyTrain"
PRINTED OUTPUT

Question 2(b).

\[ w_0 = -6.062900969977932 \]
\[ w = [1.65978929, 1.71412587] \]

Question 2(c).

\[ \text{accuracy}_1 = 0.856 \]
\[ \text{accuracy}_2 = 0.856 \]
\[ \text{accuracy}_1 - \text{accuracy}_2 = 0.0 \]

/Users/anthonybonner/anaconda2/lib/python2.7/site-packages/sklearn/linear_model/logistic.py:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.

FutureWarning)

Question 2(g).

For \( P(C=1|x) = 0.6 \)
\[ \text{Precision} = 0.756965581830953 \]
\[ \text{Recall} = 0.6466 \]

For \( P(C=1|x) = 0.5 \)
\[ \text{Precision} = 0.7450445535551918 \]
\[ \text{Recall} = 0.8194 \]

For \( P(C=1|x) = 0.05 \)
\[ \text{Precision} = 0.47966231772831924 \]
\[ \text{Recall} = 1.0 \]

Question 4(a).

\[ \text{Test accuracy} = 0.8653333333333333 \]

Question 4(c).

\[ \text{Test accuracy} = 0.8240666666666666 \]
Question 4(d).

Test accuracy = 0.902883333333334

Question 4(e).

accuracy4e = 0.8654
accuracy4a = 0.865333333333333
accuracy4e - accuracy4a = 6.66666666665932e-05

Question 5(b).

/Users/anthonybonner/anaconda2/lib/python2.7/site-packages/sklearn/
discriminant_analysis.py:692: UserWarning: Variables are collinear
  warnings.warn("Variables are collinear")
fitting time = 6.32157492638
Training accuracy = 0.1722166666666666
Test accuracy = 0.1704

Question 5(c).

fitting time = 0.620223999023
Training accuracy = 0.5649
Test accuracy = 0.5558

Question 5(e).

Full Gaussian Bayes classifier:
fitting time = 6.35286998749
Training accuracy = 0.993316666666666
Test accuracy = 0.9469

Gaussian naive-Bayes classifier:
fitting time = 0.974528074265
Training accuracy = 0.824466666666667
Test accuracy = 0.8136

Question 5(f).

Full Gaussian Bayes classifier:
fitting time = 1.63520407677
Training accuracy = 1.0  
Test accuracy = 0.1723

Gaussian naive-Bayes classifier:  
fitting time = 0.0772061347961 
Training accuracy = 0.8398333333333333  
Test accuracy = 0.8175

Question 5(h).

----------
Training accuracy = 0.8396666666666667  
Test accuracy = 0.8175

Difference in training accuracies = -0.000166666666666483  
Difference in test accuracies = 0.0
Question 1(c): sample cluster data
Question 2(d): training data and decision boundary
Question 2(e): three contours
Question 4(a):

Decision boundary and contours
Question 4(c):

Decision boundary and contours
Question 4(d):
Decision boundary and contours
Question 5(a): 25 random MNIST images

3 3 5 6 2
7 7 5 5 1
1 7 1 0 0
4 0 9 6 3
2 1 9 7 4
Question 5(d): 25 noisy MNIST images
Question 5(g): means for each digit class

0 1 2 3
4 5 6 7
8 9
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: ______________________________