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 the NumPy, SciPy and scikit-learn libraries. Your programs 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 plot and figure in matplotlib.pyplot useful, as well as the function numpy.linspace. 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 can use numpy.matmul(A,B). Also, whenever possible, do not use loops, which are very slow in Python. Instead, use Numpy’s vector and matrix operations, which are much faster and can be executed in parallel. If you haven’t already done so, please read the NumPy tutorial on the course web page.

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 handwriting 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. 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.

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.

More questions will be added shortly.
1. 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 `genData(mu0, mu1, N)` that generates two clusters of data, one for each class. Each cluster consists of \( N \) data points. The cluster for class 0 is centred at \( \mu_0 \), and the cluster for class 1 is centred at \( \mu_1 \). Note that \( \mu_0 \) and \( \mu_1 \) and all the data points are 2-dimensional vectors. The function should return two arrays, \( X \) and \( t \), representing data points and target values, respectively. \( X \) is a \( 2N \times 2 \) dimensional array in which each row is a data point. \( t \) is a \( 2N \)-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 the clusters as Gaussian distributions. To generate data for cluster 1, first use the function `randn` in `numpy.random` to generate a 2-dimensional Gaussian cluster of \( N \) points centred at the origin. Then add \( \mu_1 \) to each of the points. Likewise for cluster 0. You may find the function `numpy.concatenate` useful. You may also find the function `shuffle` in `sklearn.utils` useful for distributing the data randomly in the arrays.

(b) In this question, you will test your function from part (a) by displaying the data as a scatter plot. The following code does exactly this, displaying cluster 0 in red and cluster 1 in blue:

```python
import matplotlib.pyplot as plt
X, t = genData(mu0, mu1, N)
colors = np.array(['r', 'b']) # 0 is red and 1 is blue
plt.figure()
plt.scatter(X[:, 0], X[:, 1], color=colors[t], s=10)
```

Execute this code using \( \mu_0 = (0.5, -0.5) \), \( \mu_1 = (-0.5, 0.5) \) and \( N = 10,000 \). Of course, you will have to encode the vectors as appropriate numpy arrays. Be sure to use the parameter setting \( s=10 \) in the last line, which specifies a large dot size.

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 near the center of the figure, which is a result of distributing the data points randomly in the arrays. Title the plot “Figure 1: scatter plot of data”. To increase the detail in the plot, you should enlarge the figure (perhaps even putting it into full-screen mode) before saving it. Hand in your plot.

Note that at any point, \( x \), in Figure 1, the two clusters overlap, so that \( x \) may be
Figure 1: scatter plot of data
either red or blue. Moreover, the amount of overlap varies from point to point in the figure. For example, in the middle of the figure, \( x \) is equally likely to be red or blue, but in the top left of the figure, it is more likely to be blue, and in the bottom right, it is more likely to be red. Logistic regression models this varying probability with a sigmoid function:

\[
P(C = 1 | x) = \frac{1}{1 + e^{-z}}
\]

where \( z = w^T x + w_0 \). (In fact, when the two clusters are Gaussian, as they are in our case, this equation is provably correct.)

Try generating and displaying other data sets (but do not hand in the figures). For example, if \( \mu_0 = (2.0, -2.0) \), \( \mu_1 = (-2.0, 2.0) \), you should get two clusters that barely overlap at all. Try doing this with \( N=10,000 \) and \( N=1,000 \), to see the true shape of the clusters, and with \( N=10 \), which is more typical of real data.

2. Logistic Regression

In this question, you will experiment with (binary) logistic regression on a variety of data sets generated by \texttt{genData}. You will also visualize the results as contour plots and surfaces in 3D.

The following code trains a classifier using logistic regression on data from Question 1:

```python
import sklearn.linear_model as lin
X,t = genData(mu0,mu1,N)
clf = lin.LogisticRegression()
clf.fit(X,t)
```

Here, \( \texttt{clf} \) is a Python object representing the classifier, and \( \texttt{clf.fit} \) trains the classifier (i.e., fits it to data).

To visualize the decision surface of the trained classifier, download and import the file \texttt{bonnerlib2.py} from the course web page (uncompressing it if necessary). The following code then displays a contour plot of the decision function (i.e., the function \( f(x) = P(C = 1 | x) \)) superimposed on a scatter plot of the data:

```python
import matplotlib.pyplot as plt
import bonnerlib
fig1 = plt.figure()
ax1 = plt.gca()
ax1.set_xlim(-4,4)
ax1.set_ylim(-4,4)
fig1.suptitle('Figure 2: contour plot of logistic decision function')
colors = np.array(['r','b'])
ax1.scatter(X[:, 0], X[:, 1], color=colors[t],s=1)
bonnerlib.dfContour(clf,ax1)
```
Here, `fig1` and `ax1` are Python objects representing the figure and axes, respectively. (We will need this distinction later to handle multiple subplots in multiple figures.) The middle two lines set the x and y axes to extend from -4 to 4, the second last line draws the scatter plot,¹ and the last line draws the contour plot.

Try executing this code on the data from Question 1(b). If you have done everything correctly, the result should be very similar to Figure 2. (Do not hand in the figure.) The black diagonal line in this figure is the decision boundary. The two clusters overlap equally on this line, so any point on the line is equally likely to be in either class. Formally, \( P(C = 1|x) = 0.5 \) for every point, \( x \), on the line. This is equivalent to the equation \( w^T x + w_0 = 0 \). Notice that the contour lines are closest together near the decision boundary. This is the region where the sigmoid function changes most rapidly. Also notice that there are no contours in the bottom right and top left of the figure, just a large region of red and a large region of blue, respectively. These are the regions where the sigmoid function is almost flat: it is almost equal to 0 in the bottom right, and almost equal to 1 in the top left.

The sigmoidal nature of the decision function can be seen more clearly in Figure 3, which shows a surface plot of the function in 3 dimensions. (A 2D contour plot is displayed below the 3D surface for comparison purposes.) This figure is generated by the code below. Try executing the code (but do not hand in the figure).

```python
fig2 = plt.figure()
ax2 = fig2.gca(projection='3d')
ax2.set_xlim(-9,6)
ax2.set_ylim(-6,9)
fig2.suptitle('Figure 3: surface plot of logistic decision function')
bonnerlib2.df3D(clf,ax2)
```

**What to do:** Write Python programs to carry out the following tasks, adapting the above code fragments as necessary:

(a) Write a Python function `logregDemo(N,betaList)` that demonstrates the effect of logistic regression on a variety of data sets. Here, \( N \) is a positive integer, and `betaList` is a list of real numbers, \([\beta_1, \beta_2, ..., \beta_K]\). For each value \( \beta_k \) in `betaList`, your function should do the following:

i. Use `genData` to generate a data set consisting of two clusters with \( N \) data points each. The clusters should be centered at \( \beta_k\mu_0 \) and \( \beta_k\mu_1 \), where \( \mu_0 = (2, -2) \) and \( \mu_1 = (-2, 2) \).

ii. Train a logistic regression classifier on the data set.

iii. Display a contour plot of the decision function for the classifier superimposed on a scatter plot of the data (as in Figure 2). Set the limits of the x and y axes to be -6 and 6. Use a very small dot size of \( s=0.1 \) in the scatter plot.

¹Unlike Figure 1, the parameter setting \( s=1 \) used here specifies a small dot size in the scatter plot.
Figure 2:

Figure 2: contour plot of logistic decision function
Figure 3: 
Figure 3: surface plot of logistic decision function
iv. Display a 3D surface plot of the decision function (with a 2D contour plot below the surface, as in Figure 3). Set the limits of the x axis to be -9 and 6, and the limits of the y axis to be -6 and 9 (as in Figure 3).

In addition, all the contour plots should be arranged on a square grid in a single figure. Likewise, all the surface plots should be arranged on a square grid is a single figure. (i.e., Two figures in all). The top-left plot of each grid should be for the first element of betaList. As you progress through betaList, you should move through the grid from left to right and top to bottom. The two grids should be chosen to be just large enough to hold all the plots. (e.g., if betaList has 7 values, then you will generate 7 contour plots and 7 surface plots, so you will need two 3x3 grids to display them all.) You may find the function numpy.ceil useful.

As in Assignment 1, each plot within a grid is called a subplot. Because you are generating two grids simultaneously, you need some way to specify which grid a subplot is to be added to. One way to do this is to use explicit objects for figures and axes, as in the code fragments above. In Assignment 1, you used the command subplot(m,m,k) to add the kth subplot to an mxm grid. Here, you will use ax = fig.add_subplot(m,m,k) instead, where fig is the figure object to which the subplot is to be added (created by plt.figure), and ax is an axes object for the newly-created subplot. To create a subplot with 3-dimensional axes for a surface plot, you will need to write ax = fig.add_subplot(m,m,k,projection='3d').

(b) Execute logregDemo using N=10,000 and betaList=[0.1, 0.2, 0.5, 1.0]. Hand in the two figures this produces. Note that as β increases, the two clusters overlap less and less. Using the contour plots and the surface plots, describe the changes in the decision function as the amount of overlap in the two clusters changes. Provide an intuitive (i.e., non-mathematical) explanation for why these changes occur.

3. In this question, you will use logistic regression to classify images of handwritten digits.

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 command below in Python 2.x. (’rb’ opens the file for reading in binary.) If you have any difficulty opening the file, try downloading the uncompressed version.

```python
with open('mnist.pickle', 'rb') as f:
    data = pickle.load(f)
```

The variable data will now contain a Python dictionary with two keys, called training and testing. data[‘training’] contains a tuple with 10 fields, numbered 0 through 9. Likewise for data[‘testing’]. Each field contains a 2-dimensional array having 784 columns. Each row of each array represents a hand-written digit. For example, each row of the array in field 7 represents the digit 7 (i.e., each row of data[‘training’][7],
and each row of \texttt{data['testing'][7]}. Each training array represents about 6,000 digits, and each testing array represents about 1,000 digits.

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).

To view a digit, you must first convert it to a \(28 \times 28\) array using the function \texttt{numpy.reshape}. You can check the dimensions of a NumPy array by using the function \texttt{numpy.shape}. (Alternatively, given a NumPy array, \(A\), you can use \texttt{A.shape} and \texttt{A.reshape}.) To display a 2-dimensional array as an image, you can use the function \texttt{imshow} in \texttt{matplotlib.pyplot}. To see an image in black-and-white, add the keyword argument \texttt{cmap='Greys'} to \texttt{imshow}. To remove the smoothing and see the 784 pixels clearly, add the keyword argument \texttt{interpolation='nearest'}. Try displaying a few digits.

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

(a) Suppose \(D\) is an array in which each row represents an image. The array need not have 784 columns, but you can assume the image is square; that is, each image has \(m \times m\) pixels, so the array has \(m^2\) columns, for some \(m\). Write a Python program \texttt{displaySample(N,D)} that displays \(N\) distinct and randomly chosen rows of array \(D\) as images arranged in a roughly square grid in a single figure. The grid should be just large enough to hold the \(N\) images. Use the function \texttt{resample} in \texttt{sklearn.utils} to choose the random images, setting the keyword argument \texttt{replace=False} to sample without replacement (and thus get distinct images). In addition, use the function \texttt{matplotlib.pyplot.axis} to turn off the axes in each image. You may also find the functions \texttt{numpy.sqrt} and \texttt{numpy.ceil} useful. The images should be in black-and-white and should not use any smoothing. Use your program to display 14 random images of the digit 5 from the MNIST training data. Title the figure ‘Figure 4: random MNIST images of the digit 5’. You may find the function \texttt{suptitle} in \texttt{matplotlib.pyplot} useful for this.

(b) Use your function \texttt{displaySample} to display a sample of 23 random images from all the MNIST training data. Title the figure ‘Figure 5: random sample of MNIST training images. Figure 4 shows how it should look. You may find the function \texttt{numpy.vstack} useful.

(c) The training and test data read in from the file \texttt{mnist.pickle} is not in the form required by the functions for training and testing classifiers in scikit-learn. In particular, they are not in the form required for the logistic-regression classifier. Write a function \texttt{flatten(data)} that transforms the data into the correct form. Here, \texttt{data} is a list or tuple of data matrices, one for each class, like the MNIST training and testing data. The function should return a pair \(X,t\), where \(X\) is a single data matrix containing all of the data points from all classes, and \(t\) is a corresponding vector of class labels. That is, if row \(i\) of \(X\) is a data point in class
c, then $\mathbf{t}[i] = c$. If there are $K$ classes, then $c$ is an integer between 0 and $K - 1$, inclusive.
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:  