University of Toronto Mississauga
Department of Mathematical and Computational Sciences
CSC 411 - Machine Learning and Data Mining, Fall 2017

Assignment 2

Due date: Monday November 13, 11:59pm.
No late assignments will be accepted.

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

All computer problems are to be done in Python with 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 hand-writing is fine), and a scanned, signed copy of the cover sheet at the end of the assignment. Be sure to indicate clearly which question(s) each program and piece of output refers to. 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. 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.

No more questions will be added.

Use the solutions to Assignment 1 as a guide/model for how to present your solutions to Assignment 2.

1. (7 points total) *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) (6 points) 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 `mu0`, and the cluster for class 1 is centred at `mu1`. Note that `mu0` and `mu1` 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 `mu1` 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) (1 point) 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:

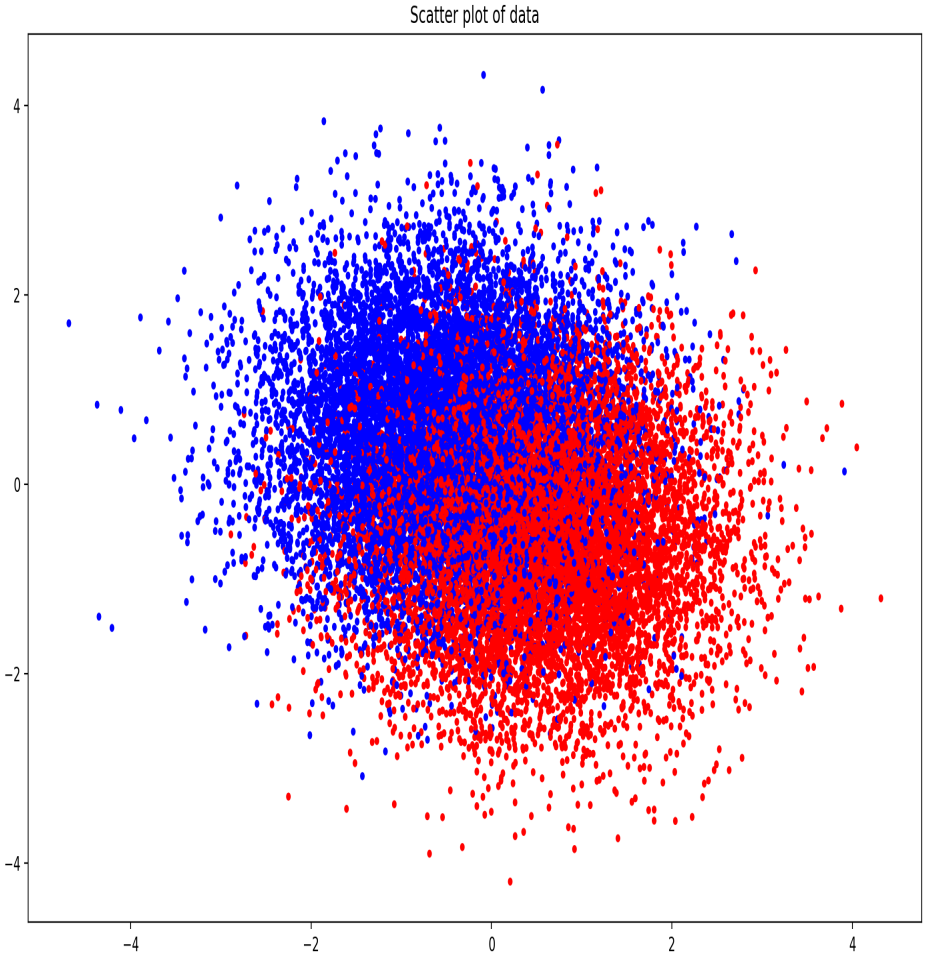
```
import matplotlib.pyplot as plt
X,t = genData(mu0,mu1,N)
colors = np.array(['r','b'])    # 0 is red and 1 is blue
plt.figure()
plt.scatter(X[:, 0], X[:, 1], color=colors[t],s=10)
```

   Execute this code using `mu0 = (0.5,-0.5)`, `mu1 = (-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:



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:

$$P(C = 1|x) \;=\; \frac{1}{1 + e^{-z}} \tag{1}$$

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 `mu0 = (2.0,-2.0)`, `mu1 = (-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. (16 points total) *Binary Logistic Regression.*

   In this question, you will experiment with (binary) logistic regression on a variety of data sets generated by `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 and computes its accuracy:

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

   Here, `clf` is a Python object representing the classifier, `clf.fit` trains the classifier (*i.e.*, fits it to data), and `clf.score` computes its accuracy on the data set `(X,t)`. Accuracy is the proportion of correct classifications. (*i.e.*, It is a number between 0 and 1, where 1 is perfect accuracy.) Test accuracy can be computed by passing test data to `clf.score` instead of training data.

   To visualize the decision surface of the trained classifier, download and import the file `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:

```
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('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).

```
fig2 = plt.figure()
ax2 = fig2.gca(projection='3d')
ax2.set_xlim(-9,6)
ax2.set_ylim(-6,9)
fig2.suptitle('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) (10 points) Write a Python function `logregDemo(N,betaList)` that demonstrates the effect of logistic regression on a variety of data sets in which the clusters have different amounts of overlap. Here, `N` is a positive integer, and `betaList` is a list of real numbers, $[\beta_1, \beta_2, ..., \beta_K]$, specifying the amount of overlap (where lower values imply more overlap). Specifically, for each value, β_k, in `betaList`, your function should do the following:

[1] Unlike Figure 1, the parameter setting `s=1` used here specifies a small dot size in the scatter plot.

Figure 2:

Contour plot of logistic decision function

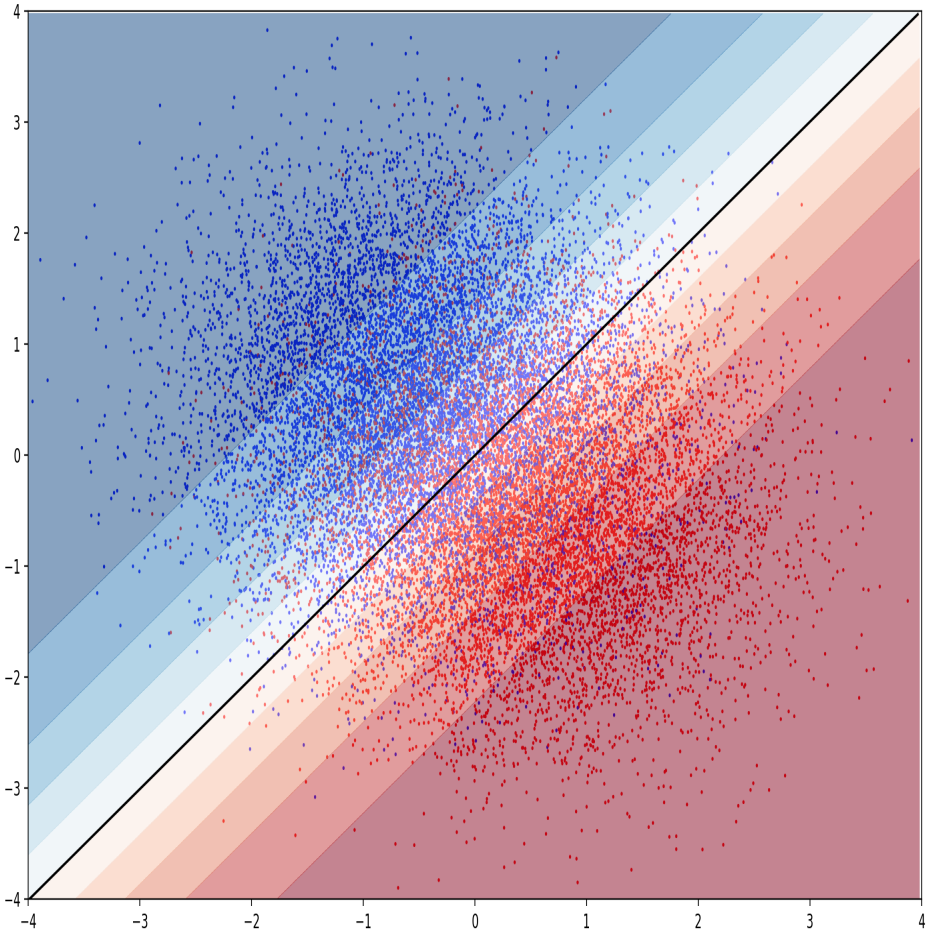
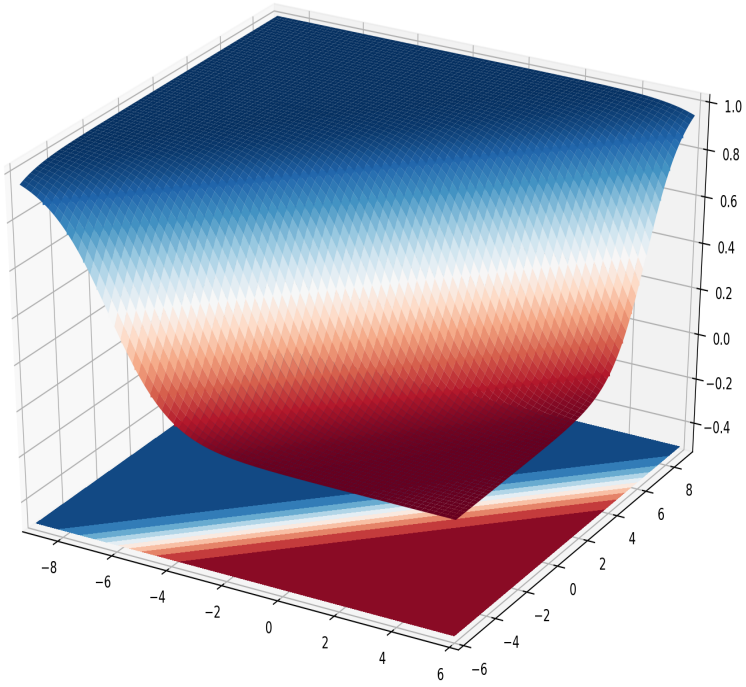


Figure 3:

Surface plot of logistic decision function



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. Compute and print out the accuracy of the classifer on the training set.

iv. 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.

v. 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).

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. (Two figures in all). The figure with the grid of contour plots should be titled, "Figure 2: contour plots of logistic decision functions," and the figure with the grid of surface plots should be titled, "Figure 3: surface plots of logistic decision functions." 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 k^{th} 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')`.

Hand in all the plots and printed values. It should be clear what each piece of output refers to.

(b) (1 point) 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.

(c) (5 points) 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. Also describe the changes in the accuracy of the classifier. Provide an intuitive (*i.e.*, non-mathematical) explanation for why all these changes occur.

3. (28 points total) *Multi-class Logistic Regression.*

In this question, you will use logistic regression to classify images of handwritten digits. There are ten different digits (0 to 9), so you will be using multi-class logistic regression.

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 one of the commands below, the first for Python 2.x, and the second for Python 3.x. (`'rb'` opens the file for reading in binary.) If you have any difficulty opening the file, try downloading the uncompressed version.

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

with open('mnist.pickle','rb') as f:
    data = pickle.load(f,encoding='latin1')
```

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 `data['testing'][7]`). Each training array represents about 6,000 digits, and each testing array represents about 1,000 digits. There are a total of 60,000 training images and 10,000 test images.

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 28×28 array using the function `numpy.reshape`. You can check the dimensions of a NumPy array by using the function `numpy.shape`. (Alternatively, given a NumPy array, `A`, you can use `A.shape` and `A.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. For comparison, try printing them as vectors (*i.e.*, using `print`).

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

(a) (6 points) 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 `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

Figure 4:

should be just large enough to hold the `N` images. Use the function `resample` in `sklearn.utils` to choose the random images, setting the keyword argument `replace=False` to sample without replacement (and thus get distinct images). In addition, use the function `matplotlib.pyplot.axis` to turn off the axes in each image. You may also find the functions `numpy.sqrt` and `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 `suptitle` in `matplotlib.pyplot` useful for this.

(b) (2 points) Use your function `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 `numpy.vstack` useful.

(c) (6 points) The training and test data read in from the file `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 `flatten(data)` that transforms the data into the correct form. Here, `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 (target values). For example, if row

`i` of `X` is an image of the digit 7, then `t[i] = 7`. If `data` is the MNIST training data, then `X` will have 60,000 rows and 784 columns, and `t` will have 60,000 entries.

(d) (4 points) Train a logistic-regression classifier on the MNIST training data and print out the training accuracy and the test accuracy. The following code defines the correct kind of multi-class classifier:

```
import sklearn.linear_model as lin
clf = lin.LogisticRegression(multi_class='multinomial',solver='lbfgs')
```

In particiular, the resulting classifier, `clf`, will *not* use a one-vs-one or one-vs-the-rest scheme. Instead, it will use a separate discriminant function for each class, a softmax function to convert the discriminant values to probabilities, and a cross-entropy loss function (as described in class). You should fit the classifier to the training data (but *not* to the test data), and then compute its accuracy (score) on the training and on the test data. If you have done everything correctly, the test accuracy should be a little less than the training accuracy, and both should be a little bit bigger than 0.9. Training the classifier may take a little time (16 seconds on my laptop).

(e) (5 points) If `clf` is a trained classifier, then the method `clf.predict` makes predictions for data samples. Specifically, if `X` is a data matrix (in which each row is a data point), then `clf.predict(X)` returns a vector of predictions, that is, a vector of integers, where the i^{th} entry is the predicted class of the i^{th} data point in `X`. Your task in this question is use `displaySample` to display a random sample of 36 MNIST test images that are *incorrectly* classified by your trained classifier from part (d) (*i.e.*, whose predicted class is not the true/target class). Title the figure, "Figure 6: some missclassified images." Many of the images should appear to be sloppily-written digits. (You will have no problem identifying most of the missclassified images, which shows that there is considerable room for improvement. More-complex methods, like neural nets, make far fewer mistakes.)

(f) (5 points) If `X` is a data matrix, then `clf.predict_proba(X)` returns a matrix, `P`, of predicted probabilities. That is, the i^{th} row of `P` contains the predicted probabilities for the i^{th} row of `X`. Specifically, if x is the data sample in the i^{th} row of `X`, and p is the vector in the i^{th} row of `P`, then $p[k]$ is the predicted probability that x belongs to class k. The class, k, with the largest probability is the prediction returned by `clf.predict`. The higher this maximum probability is, the greater the confidence of the prediction. Your task in this question is to display the MNIST test images with the *least* confident predictions. To do this, first compute the maximum predicted probability for each MNIST test image. Then use `displaySample` to display the 36 images with the lowest maximum probability. (Many of the images should be of poor quality or sloppily-written.) Title the figure, "Figure 7: images with the least confident predictions." You may find the numpy functions `max` and `argsort` useful.

4. (20 points total) *Logistic Regression: Theory.*

Recall that for binary logistic regression, the loss function is given by the following formula:

$$l(w, w_0) \;=\; -\sum_n t^{(n)} \log P(C=1|x^{(n)}) - \sum_n (1-t^{(n)}) \log P(C=0|x^{(n)})$$

where $P(C=0|x) = 1/(1+e^{-z})$ and $z = w^T x + w_0$. Each $x^{(n)}$ is a training point,[2] and each $t^{(n)}$ is a target value of 0 or 1. On the midterm test, you showed that this loss function can be simplified to the following:

$$l(w, w_0) \;=\; \sum_n t^{(n)} z^{(n)} + \sum_n \log\left[1 + e^{-z^{(n)}}\right]$$

(a) (10 points) Letting $y^{(n)} = P(C=1|x^{(n)})$, prove that for $j \geq 1$,

$$\frac{\partial l(w, w_0)}{\partial w_j} \;=\; \sum_n [t^{(n)} - y^{(n)}] x_j^{(n)}$$

where $w = (w_1, w_2, ..., w_M)$ and $x^{(n)} = [x_1^{(n)}, x_2^{(n)}, ..., x_M^{(n)}]$, both treated as column vectors.

(b) (2 points) Without going through a separate proof, argue that

$$\frac{\partial l(w, w_0)}{\partial w_0} \;=\; \sum_n [t^{(n)} - y^{(n)}]$$

(c) (5 points) Let t and y be the vectors $[t^{(1)}, t^{(2)}, ..., t^{(N)}]$ and $[y^{(1)}, y^{(2)}, ..., y^{(N)}]$, respectively, and let X be the data matrix whose n^{th} row is the vector $x^{(n)}$. Treating t, y and $\partial l/\partial w$ as column vectors, prove that

$$\frac{\partial l(w, w_0)}{\partial w} \;=\; X^T(t - y)$$

(This is the formula one would use in gradient descent.)

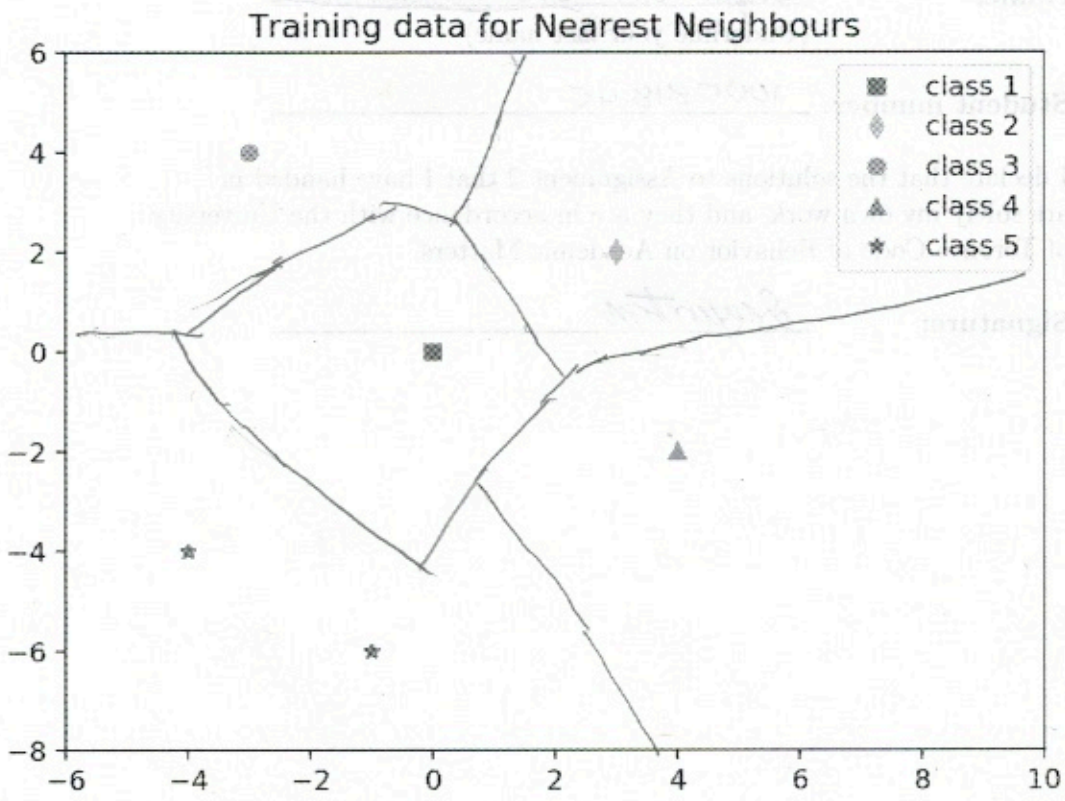
(d) (3 points) Write a Python program that uses binary logistic regression and the MNIST data to train and test a classifier to distinguish images of handwritten 2's from images of handwritten 3's. Report the test accuracy. If your `flatten` function in Question 3(c) is general enough, you will be able to use it here to prepare the data.

5. (5 points) *Nearest Neighbours.*

Figure 5 shows a scatter plot of six data points belonging to five different classes (as indicated by the five different marker styles). By hand (*i.e.*, with pencil and paper), draw the decision boundary for this training set when using 1NN.

[2] For the cluster data of Question 1, each $x^{(n)}$ would have 2 dimensions. For the MNIST data, each $x^{(n)}$ would have 784 dimensions, as in part (d) of this question.

Figure 5:


Training data for Nearest Neighbours
class 1
class 2
class 3
class 4
class 5
6
4
2
0
−2
−4
−6
−8
−6
−4
−2
0
2
4
6
8
10

6. (10 points) *Decision Trees.*

   We want to predict whether a potential buyer will like a car or not. By hand (*i.e.*, with pencil and paper), construct a 2-level decision tree[3] based on the training examples in the table below. Be sure to show the value of the information gain for each candidate attribute at each step in the growing tree. Do not split a node if its entropy is 0. Also, do not split the grandchildren of the root (*i.e.*, nodes that are two levels below the root).

Example	Make	Type	Colour	Cost	LikeCar
1	Toyota	Sedan	Blue	High	Yes
2	Toyota	Sedan	White	Low	No
3	Ford	SUV	Blue	High	No
4	Ford	Sedan	Blue	Low	Yes
5	Toyota	Sedan	White	Low	No
6	Ford	Sedan	White	Low	Yes

[3] *i.e.*, The tree should have a root with children and grandchildren.

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: Sayantan Chattopadhyay
(Underline your last name)

Student number: 1000818145

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: Sayantan