Lab 4
CO902 – Probabilistic and statistical inference – 2012-13 Term 2
Lecturer: Tom Nichols
Grayscale Handwritten Digit Identification

Develop a Naive Bayesian classifier for continuous (Gaussian) data of handwritten digits. Unlike my example in lecture, these digits have grayscale values between 0 and 256.andwritten Digits dataset, contained in the file digits.mat (on the website). In the digits.mat file you will find a two variables, `digit_img`, and `digit_lab`. The `digit_img` variable is a 256 × 2000 matrix, where the first dimension is "space" unwrapped (i.e. the images of handwritten digits are 16×16 = 256 arrays of pixels), and the second dimension are the 2000 cases. The `digit_lab` variable is a length 2000 label vector telling you the true identity of the digit, one of 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 (10 means "zero:").

1. Explore the data. Plotting a single case `plot(digit_img(:,1))` won't be so interesting (try it!), as it will just show the "unwrapped" images. View a few images using the Matlab function "reshape", which changes the dimensions of a vector/matrix/array (so long as the total number of elements remains constant). Try copying and pasting this snippet (get PDF from webpage) a couple times

   ```
   colormap gray
   i = randi(2000,1); % A random case
   imagesc(reshape(digit_img(:,i),[16 16])); digit_lab(i)
   ```

   Look up the help on each function here if you're not sure what is going on here.

2. Write a function called "train_digit_classifier" that takes a matrix (256 × *n*) and a label vector (length *n*) and returns the parameters needed to define a Naive Bayesian classifier. For example, to train on the first 100 cases

   ```
   [ccMean ccVar] = ...
    train_digit_classifier(digit_img(:,1:100),digit_lab(1:100));
   ```

   where `ccMean` and `ccVar` are the class conditional mean and variance of each variable for each class, i.e. they should be matrices of size 256 x 10 (i.e. mean and variance for each pixel values, for each digit). Be sure to look at (make images, like above) of the class conditional means and variances and make sure they make sense.

   This code snippet is nice for visualizing the parameters and the locations of constant voxels:

   ```
   for i=1:10
     subplot(3,10,i);    imagesc(reshape(ccMean(:,i),     [16 16]),[0 256]); axis image
     subplot(3,10,i+10); imagesc(reshape(sqrt(ccVar(:,i)),[16 16]),[0 150]); axis image
     subplot(3,10,i+20); imagesc(reshape(ccVar(:,i)==0,   [16 16]),[0 1]);   axis image
   end
   ```

3. Write a function to "test_digit_classifier" which takes another 3-way array and then returns a matrix giving the predicted class and the overall accuracy. For example, to test on the next 100 cases, the usage would be like

   ```
   [Accu, Class] = ...
     test_digit_classifier(ccMean,ccVar,...
     digit_img(:,101:200), digit_lab(101:200));
   ```

   where `Accu` is the overall accuracy (fraction correctly classified) and `Class` is a length

100 vector with the predicted class for each case.

4. Depending on the speed of your code, train and classify as much as possible. What sort of accuracy do you get? Does the accuracy improve if you use "feature" selection? What are the important pixels? (They must be the same for all k, because you of course don't know k when classifying!) What if you assume common variance over the groups?