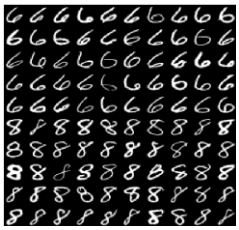
EC-350 Artificial Intelligence and Decision Support Systems
Assignment 3
Handwritten Digit Recognition



Deadline: Wed 27th Dec, 2017

Download the folder `assignment1`. Go in there, and you will find several files. You should see `usps_main.mat`, and `usps_benchmark.mat`. These are samples of some US Postal Service handwritten digit data – the `usps_main` set contains 500 examples of each of the digits 0-9 (so contains 5,000 digits totally). The `usps_benchmark` set is a different 500.

The aim of the assignment is to correctly recognise the "benchmark" examples of digits 3, 6, and 8. You should use only data from the "main" set for the training of your KNN rule.

You are provided with an implementation of a K-nearest neighbor classifier. At the Matlab command line, do help knearest to find out how to use it. You also have the following utility functions:

```
showdigit - displays image for a single digit
getonedigit - retrieves pixels for a single digit
showdata - displays a group of digit images
crossfold - partitions the data
shufflerows - shuffles the rows
extractfeatures - transforms the raw pixels
knearest – implements the k-nearest neighbor classifier
```

Use the help command again to find out how they work – be sure to know what arguments they take, and what they return.

You load the digit data into memory with `load usps_main.mat`. After that, you can use the various functions listed above. As a starter, try the following:

```
getonedigit(3, 123, maindata)
```

Notice that we passed the variable `maindata` as an argument – this is the variable that should have been loaded when you typed `load usps_main.mat`. Now pass the output of that function to the function `showdigit`. A number '3' should be displayed on the screen. This is the 123rd example of a number '3'. There are totally 500 examples of each of the digits 0-9. Try others for yourself.

Since Matlab indexes arrays from 1 then to get examples of the digit '0', you use array index 10, as follows:

```
showdigit( getonedigit(10, 312, main) )
```

Take note, all that the `getonedigit` function is doing is pulling out columns of the variable `main`. You could do this equally well yourself, and probably write something to make it more flexible. Look at the code by typing `edit getonedigit`.

PART 1: Tasks to be Performed

1. Write a script that constructs a training dataset of just '3' and '8' digits. To start, pick 100 of each randomly. Your matrix should end up as 2D, 200 rows by 256 columns. Remember to include the true label for each digit, in another array, called `labels` (or whatever you want).

2. Use the KNN rule to classify each of the digits in your training set, and report the training accuracy. Plot a graph to display the training accuracy as you vary K from 1 to 20.

3. Now break your training set randomly into 2 equal parts, one part you will use for training, and one part for testing. Plot a testing accuracy graph, again varying K.

4. Again perform above task 3 by repeating the random split of data into training and testing and then reclassify. Do you get the same behaviour?

5. Plot the average and standard deviation of both testing accuracies achieved in task 3 and 4 as error bars, for each value of K. Remember all graphs should have axis labels and a title. If you don't know what Matlab commands to use, try Googling.

6. When you are done with all this, extend the above to load and predict digits '3', '6', and '8'. You can visualise your classifications using the **`showdata`** function provided.

PART 2: Feature Extractors for Digit Recognition

Take a copy of the function **`extractfeatures`**, and rename it **`extractmyfeatures`**. Remember the name of the function has to be the same as the name of the file containing it.

The **`extractfeatures`** function transforms the 256 pixels of any image into a **lower dimensional representation,** called a set of **features**. The features are meant to "summarise" the pixel information, but in less dimensions. The current function sums up the pixel values in each column – giving a feature set of size 16, since the images are all 16x16. Another might be to use all the odd numbered pixels, giving a feature set of size 128. Each of these is likely to give us different **accuracy** when we try to classify the digits.

Use your imagination to design new feature extractors – which preserve the "information" in the images, but reduces the number of dimensions necessary. Is there an observable trade-off between how many features you use, and the accuracy you get? It is clear that using **less features** for the KNN rule would be better in both time and memory requirements, but do you get a high accuracy? You should calculate and report the accuracy using your features. Note that the accuracy of a KNN classifier is defined as follows:

Accuracy = number of correctly classified test instances / total number of test instances

For example, if you are given a test dataset of 100 instances and your code correctly predicts labels (i.e., predicted labels are as same as their true labels) for 85 instances, the accuracy is 85/100 = 0.85.

Remember, you should only evaluate your accuracy on the 3, 6, and 8 digits within the benchmark set. This means the total number of examples in your benchmark set should be 1500. That is: five hundred 3's, five hundred 6's and five hundred 8's.
The other digits should be discarded, or you can try to classify them just for fun.

Remember, you can use the function `showdata` to visualize your classifications. White boxes are placed around digits that you have predicted incorrectly. What do you notice? Are you always misclassifying examples of the '8' digits? Or the '3' digits?

The simple accuracy measure we adopted does not tell us **which digits we commonly mis-predict.** This information obviously might be useful to know – we wouldn't want to use a KNN rule that can never recognize the number 8. A far more useful evaluation measure is to look at the **confusion matrix.** Build a **confusion matrix** for your classifications on the benchmark set.

Deliverables

A print out of your report, including your code + figures, graphs, tables etc. + maximum one page description.

Marks will be allocated roughly on the basis of:

- Rigorous experimentation,
- How informative/well presented your graphs are,
- Grammar, ease of reading.