COMP24111 – Ex 2

Handwritten Digit Recognition

Deadline: See website.
Extension policy: NO EXTENSIONS (EVEN ON REQUEST)

You need to do 3 things:

1) Download a zip.
2) Read the help text and experiment with the utility functions I have provided.

then (and ONLY then) …

3) Do the lab exercises.

Download a zipfile:

http://www.cs.man.ac.uk/~gbrown/comp24111code.zip

In this zip file you will find 5 files with extension ".m":

```
knearest.m
extractfeatures.m
showdata.m
showdigit.m
shufflerows.m
```

These are the utility functions that you will use in this miniproject. To get help on how these scripts work, use the help facility, e.g. **`help knearest.`** Use the help command to find out how they work, or look at the code itself – be sure to know what arguments they take, and what they return, so you **don't bother the demonstrators** with something that you could solve by just typing: **`help showdigit`**

You will also find a single file with extension ".mat":

```
postaldata.mat
```

This is a datafile, containing the data you will use for the project. If you have unzipped this, start Matlab. Then, use the Matlab command line (not the Unix command line) to navigate to the appropriate diretory where these files are. Now type :

```
load postaldata
```

This will have loaded two variables into memory – **`data`** and **`labels`**. These are samples of some US Postal Service handwritten digit data – the **`data`** matrix, 5000 rows by 256 columns, and contains the actual images, and the **`labels`** variable contains integers saying what digit is contained in each image.

PART 0 – Preparation

As a starter, try the following:

```
load postaldata
d = data(1123,:);
showdigit(d);
```

This has pulled out the 1123rd row, and all columns from the matrix. A number '3' should be displayed be on the screen. There are totally 500 examples of each of the digits 0-9. Try others for yourself.

Since Matlab indexes arrays from 1 (I hope you all realised that by now) then to get examples of the digit '0', you use array index 10, as follows:

```
showdigit( data(4950,:) );
```

PART 1 – Training and Testing

a) 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 **`mylabels`** (or whatever you want).

b) Use the KNN rule to classify each of the digits in your training set, and report the accuracy. Plot a graph to display the accuracy as you vary K (i.e. K on the x-axis, and accuracy on the y-axis). What does k=1 mean? What does it mean as k gets bigger?

c) Now, plot a *testing* accuracy graph, again varying K. Use your original 200 training datapoints, and pick another 200 appropriate digits for testing. This of course means you need to know the difference between training accuracy and testing accuracy, which you saw in the notes/lectures.

d) Repeat the process for (c) many times. Do you get the same behaviour? If not, why not? Plot an average accuracy, and standard deviation as error bars. Remember all graphs should have axis labels and a title. If you don't know what Matlab commands to use, try Googling to find out….

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

PART 2 – Feature Extractors for Digit Recognition

Edit the file **`extractfeatures`**. Remember Matlab is similar to Java in that 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 me different accuracy when I 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. You should evaluate a "score" for yourself as:

$$score = \frac{\text{accuracy on benchmark set}}{\text{number of features used}}$$

For example, if I got 90% accuracy, using the 256 original pixel values as features, my score is:

$$score = \frac{90}{256} \approx 0.351$$

Obviously, if I used only 16 features, and still got 90% accuracy, I'd have a higher score, about 5.6. Maybe for a bit of fun, compare your score to your neighbours. However, remember your score does not in any way translate to your grade on this exercise – it's JUST FOR FUN.

Remember, you can use the function **`showdata`** to visualise 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 recognise the number 8. A far more useful evaluation measure is to look at the **confusion matrix:**

`http://en.wikipedia.org/wiki/Confusion_matrix`

Try building a **confusion matrix** for your classifications on the benchmark set.

PART 3 – Something a bit more challenging… Perceptrons

Implement a Perceptron, as described in the lecture**.** Remember, that a Perceptron can only do **two-class problems**, so you should downside the dataset to just recognising digits '6' vs. '8'…. or '3' vs. '8'… or '3' vs. '6'. Compare the performance to a 2-class KNN, in terms of both time and space complexity. How long does training take? How much memory is necessary? What testing accuracy do you achieve?

PART 4 – Bonus marks

Additional, bonus marks are available for truly exceptional students: those able to show observations and knowledge that was not explicitly supplied in lectures. To restate - to obtain marks in this category you should **have completed fully parts 1-3 and then shown evidence of additional learning outside the supplied lecture notes.** Examples of things you could do: learn about other forms of evaluation than just a confusion matrix, or other feature extraction routines, or other classifiers than just the ones we learnt in the lectures. Simply running code you have downloaded from somewhere else will not gain any marks.

DELIVERABLES

By the deadline, you should submit two files, using the **`submit`** command as you did last year:

```
ex2code.zip       - all your .m files, zipped up
ex2report.pdf     - your report, as detailed below
```

A report should be submitted, on **a single sheet of A4 (two sided is ok)** as a PDF file, but also printed out to show your demonstrator.

REMEMBER to write your NAME and STUDENT ID on it!

You should give a full description of the feature extraction you designed for Part 2, and any graphs that you think represent your achievements in Parts 2 (and Part 3 if you did it). Take note, we are not interested in the details of your code, what functions are called, what they return etc. This module is about algorithms, and is indifferent to how you implement them.

There is no specific format – marks will be allocated roughly on the basis of:

- rigorous experimentation,
- knowledge displayed when talking to the demonstrator,
- imagination in Part 2 above,
- how informative / well presented your graphs are,
- grammar, ease of reading.

The lab is marked out of 20:

Part 1 – Training and Testing	**8 marks**
Part 2 – Feature Extractors	**4 marks**
Part 3 – Perceptrons	**4 marks**
Part 4 – Bonus	**4 marks**

Remember – a mark of 14/20 constitutes a first class mark, 70%.

Bonus marks will only be awarded if you show evidence of reading and understanding outside the lecture notes.