COMP5850/8260 AI Systems Implementation
Class I: Exploring abstract features

The purpose of this class is to implement and explore the effectiveness of more abstract features as part of object recognition tools. Specifically, it is to consider using moments as pattern classification features and gain an understanding of their effectiveness. The proposed systems uses a Decision Tree classifier to identify binarised handwritten characters.

Preparation

The suggested test data to use is located at \\raptor.kent.ac.uk\exports\courses\comp8260_5850 under a folder call digits. Please copy this into your space on \\jupyter.kent.ac.uk\exports\home before starting.

Also, either download the skeleton Jupyter notebook for Class 1 from Moodle and upload it into your system or copy it from the \\raptor.kent.ac.uk\exports\courses\comp8260_5850 directory and do likelwise. The notebook contains some basic functions for you to work with which will form the basis for your system. It is not a complete system however.

Task 1

Examine the skeleton Jupyter notebook provided. The code should be *relatively* easy to understand but understanding code is always a challenging task, Therefore, if, at any time, there is a section of code which is unclear, please ask for help as it is important you understand what is happening. Firstly amend the Constant in the first cell

The code to calculate the actual moment for an image is not provided so the first task is to provide this definition. Enter this code for a non-normalised moment (e.g. m_{00}, m_{01}, m_{10} and m_{11}) n the second code cell of the notebook in place of skeleton function definition. Searching to the end of the notebook, you will find a Driver function which by default will call a function to test the values produced. If it works correctly, the output should be:

{'m00': [214.0], 'm01': [2759.0], 'm10': [1699.0], 'm11': [21463.0], 'class_id': []}

Task 2

As an introduction to exploring the effectiveness of moments, let's use a two class problem, telling the difference between handwritten "0" and "1" numerals. Let's try using non-normalised first order moments and one second order moment m_{00}, m_{01}, m_{10} and m_{11} with 10 training patterns first.

By default, the number of training patterns and the maximum values of the *p* and *q* moment parameters used are set by constants in the first cell of the notebook as values:

NUMBER_TRAINING_EXAMPLES

PMAX

QMAX

You can change these values to those you require although their defaults are 10, 2 and 2 respectively so this is not necessary for the first experiment.

Now look at the Driver function cell at the end again. At the bottom you may now comment out the **moment_tester()** function (put a # at the beginning of the line) and remove the # at the beginning of the **driver()** function call. The system may now be tested.

Note when you run the cell that the diver will prompt you for the number of training classes to explore and repeatedly prompt you for the training class and the number of samples to test for that class. The answer appears as a list. For example for ten patterns chosen from class 0, the following may appear:

```
I think the test patterns are: [0 0 0 0 0 0 0 0 0 1]
```

This means that, for the 10 examples of class 0, it thinks the first nine and 0's and the tenth is a one. As all the test patterns came from the test file for class 0, it means it correctly identified the first nine as a 0 but misclassified the final pattern as a 1.

Note that the interface is a very simple one with no error handling included and which must also be stopped manually. As you proceed through the experiments, it may prove tedious to manually re-enter the name of every class and to reset the global constants NUMBER_TRAINING_EXAMPLES, PMAX and QMAX so you may wish to edit this function to function as you desire.

So the first experiment is to try with 10 training patterns and two classes, the numerals 0 and 1

If you run it, it should print the list of pattern classifications as above. Does it work? What happens as the number of training patterns is increased?

Task 3

At this point, it is worth stopping and exploring what is actually happening. If you explore the cells in the notebook, you will find several Python **print** statements which are commented out. This is especially true for the **train_classifier()** function which has several of these. Each of these may be uncommented (remove the # at the beginning) to print out an aspect of the system. It is suggested to do these one or two at a time else the output will become confused.

For example, within **train_classifier()** it is possible to uncomment statements to, for example,

- See the actual patterns used for training
- See the feature values generated for the dataframe to be used for training
- See the decision tree produced.

This last one can be especially illuminating to see how the system is making decisions based on the contents of the training patterns. See how this tree modifies as the number of training patterns is increased. Is it making sensible inferences?

There are other options in other cells to see the test patterns or the raw feature values as they are generated.

Task 4

Next. let's explore increasing the number of classes. Suggestions are five and ten although you can try as many as you wish up to the maximum of ten. This can be achieved by typing in a number bigger than two to the number of classes question (unless you have modified this). Again explore using the same moments m_{00}, m_{01}, m_{10} and m_{11} with 10 training patterns per class initially. This time when you choose the test class, you can choose a number other than 0 or 1 up to the maximum you have chosen (e.g. 4 if you chosen five classes and 9 if you have chosen 10). You will need to re-engineer the system

slight to try unusual combinations of classes (e.g. just the even numbers) but this should not prove too challenging if you wish to do so.

How effective do you feel is the Decision Tree this time? It is best just to print out the results and see how many are correct this time as, unfortunately, if you now print out the Decision Tree, it may prove too complicated to get much of an understanding (but feel free to take a look at it if you wish). Again, increase the number of training patterns and see how the performance changes. Notice, as you test different classes, that some work better than others. Why might this be?

Task 5

Finally try increase the order of the moment. You do this by changing the Constant values PMAX and QMAX. Retry the experiments in Task 2 and 3. What are your conclusions? In this section, it may prove useful to re-engineer the driver() function to automatically cycle though all the test pattern sets and the various training options as it may prove tedious to manually enter the same numbers otherwise.

Observations

It should be apparent that, as the complexity of the problem increases, more features and training patterns are required to generate good results.

The features used are not normalised moments. The data is this case is relatively normalised but, if you have time, you may like to explore amending your code to generate normalised moments such as M_{00}, M_{01}, M_{10} and M_{11}. These would more typically be used in practice.

A Decision Tree would also not typically be used for this type of classifier. It is chosen here so we can examine it and see how its decision making changes as the number of training patterns and features varies. However, statistical classifiers such as Naïve Bayes would more likely be chosen. The probabilities produced could then be used to give confidence to the classifications.