Getting the Dataset Ready

In this lesson, we will write the code to access the files stored in MNIST database.

Before we can do anything with the data, like plotting it or training a neural network with it, we need to find a way to get at it from our Python code.

Opening a file and getting its content is really easy in Python. It's best just to show it and explain it. Have a look at the following code:

```
data_file = open("mnist_train_100.csv", 'r')
data_list = data_file.readlines()
data_file.close()
```

There are only three lines of code here. Let's talk through each one.

The first line uses a function `open()` to open a file. You can see the first parameter passed to the function is the name of the file. Actually, it is more than just the filename `mnist_train_100.csv`, it is the whole path which includes the directory the file is in. The second parameter is optional and tells Python how we want to treat the file. The `'r'` tells Python that we want to open the file for reading only, and not for writing. That way we avoid any accidents changing or deleting the data. If we tried to write to that file and change it, Python would stop us and raise an error. What's that variable `data_file`? The `open()` function creates a file handle, a reference, to that file and we've assigned it to a variable named `data_file`. Now that we've opened the file, any further actions like reading from it, are done through that handle.

The next line is simple. We use the `readlines()` function associated with the file handle `data_file`, to read all of the lines in the file into the variable data_list. The variable contains a list, where each item of the list is a string representing a line in the file. That's much more useful because we can jump to specific lines just like we would jump to specific entries in a list. So

`data_list[0]` is the first record, and `data_list[9]` is the tenth record, and so on.

By the way, you may hear people tell you not to use `readlines()` because it reads the entire file into memory. They will tell you to read one line at a time and do whatever work you need with each line, and then move onto the next line. They aren't wrong; it is more efficient to work on a line at a time, and not read the entire file into memory. However our files aren't that massive, and the code is easier if we use `readlines()`, and for us, simplicity and clarity is important as we learn Python.

The last line closes the file. It is good practice to close and clean up after using resources like files. If we don't, they remain open and can cause problems. What problems? Well, some programs might not want to write to a file that was opened elsewhere in case it led to inconsistency. It would be like two people trying to write a letter on the same piece of paper! Sometimes your computer may lock a file to prevent this kind of clash. If you don't clean up after you use files, you can have a build up of locked files. At the very least, closing a file, allows your computer to free up memory it used to hold parts of that file.

Try these codes, and see what happens when you print out elements of that list. The following shows this working.

```
#print the length of the dataset
print(len(data_list))
```

```
#print the first record
print(data_list[0])
```

You can see that the length of the list is 100. The Python `len()` function tells us how big a list is. You can also see the content of the first record

`data_list[0]`. The first number is 5 which is the label, and the rest of the 784 `data_list[0]`. The first number is 5 which is the label, and the rest of the 784 numbers are the color values for the pixels that make up the image. If you

look closely, you can tell these color values seem to range between 0 and 255. You might want to look at other records and see if that is true there too. You'll find that the color values do indeed fall within the range from 0 to 255.

We did see earlier how we might plot a rectangular array of numbers using the `imshow()` function. We want to do the same here, but we need to convert that list of comma separated numbers into a suitable array. Here are the steps to do that:

Split that long text string of comma separated values into individual values, using the commas as the place to do the splitting.

- Ignore the first value, which is the label, and take the remaining list of $28 * 28 = 784$ values and turn them into an array which has a shape of 28 rows by 28 columns.
- Plot that array!

Again, it is easiest to show the fairly simple Python code that does this and talk through the code to explain in more detail what is happening.