A6 – Classifying Handwritten Digits

Overview

- In this assignment you'll learn about a machine learning technique called *classification*.
  - You will train a learning algorithm to recognize handwritten digits. This is accomplished by providing many samples of each digit and the correct labels for the samples.
  - Then, once the algorithm has been trained, you will use it to predict the labels of samples it hasn't seen before.
- Classification is an example of *supervised* machine learning, which means that we train the algorithm by telling it the correct answers for a set of inputs.
- By contrast, in the next assignment, we'll look at a technique that uses *unsupervised* learning. We won't train the algorithm (won't tell it any answers) but ask it to find patterns in the data.

The Data

- You are provided with 1,696 samples of handwritten digits, collected from NBCC students, and captured as PNG images. (There were 1,700 samples originally – 170 per digit – but four samples were prepared incorrectly and have been excluded.)
- In any project involving data science or machine learning, the data set must be prepared so that its properties are known ahead of time.
  - For the project on linear regression, we prepared the temperature data to remove unknown temperatures and ensure that each region had at least three cities worth of data for the entire date range.
  - In this assignment, we must ensure that the samples are legible and comparable with one another.
- Thankfully, the first step of data preparation has already been done. The students who provided the samples were instructed to write with a dark colour on a pure white background and to save each image with a resolution of 32 x 32 pixels.
  - These facts are important. When we're reading the data from an image file, we know that a white pixel – rgb(255,255,255) – is part of the background.
- However, we still have some work to do to make the samples compatible with the machine learning tools we will be using. We'll do this in Part 1 below.

Software Required

You will need to install two new libraries.

1. The machine learning tools provided by Scikit-Learn. (https://scikit-learn.org/stable/)
2. The image processing tools provided by Pillow. (https://pypi.org/project/pillow/)

Part 0 – Getting Ready

- The work you'll do in this assignment is very similar to the digit classification example on Scikit-Learn's website.
- We will use the same techniques, but with our own data samples, which must be prepared. (See Part 1 below.)
- To prepare, review the code from the digit classification on Scikit-Learn's website: https://scikit-learn.org/stable/auto_examples/classification/plot_digits_classification.html

Part 1 – Creating Bitmap Files (file: a6_part1.py)

1. Write the function *create_grayscale_samples*. This function should read each digit sample file in the specified source folder and create a new file, in the target folder, in grayscale format. Keep the same file names for the new files.
    - You can simply call *convert("L")* on an Image object to convert it to a grayscale image. In a grayscale image, each pixel is simply an integer from 0 – 255 rather than a RGB triple.

2. Write the function *build_bitmaps_and_targets*. This function should read each grayscale image (each image that you created in the previous step) and create two lists:
    a. bitmaps – a Nx32x32 list where every white pixel is replaced by 0 and every non-white pixel is replaced by 1
    b. targets – a Nx1 list of digits (0 – 9) that each image represents

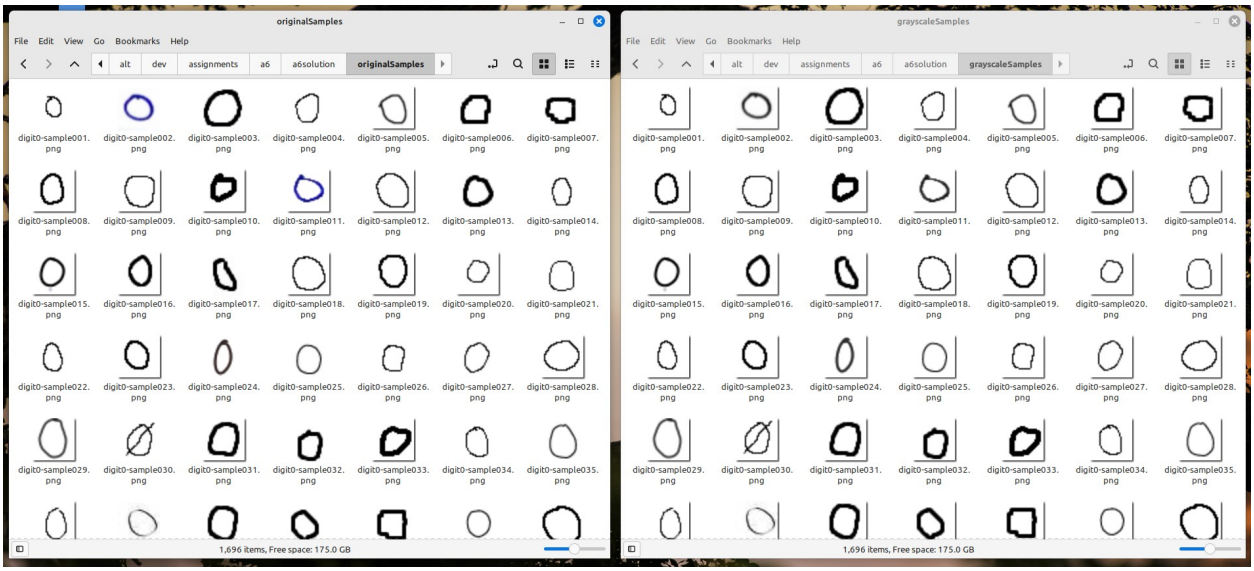
    Important: bitmaps and targets must be the same length and stored in the same order.

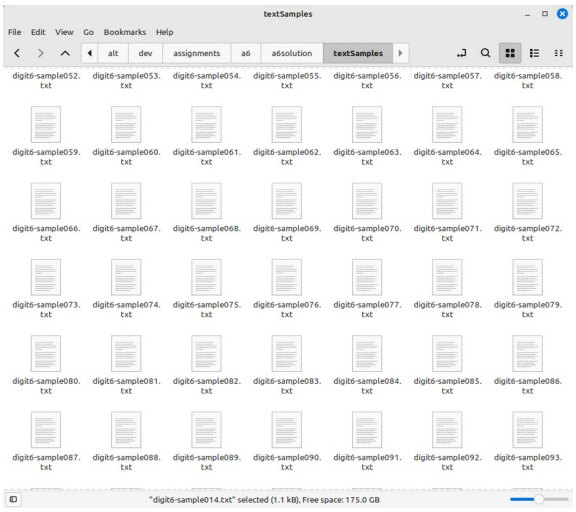
3. Write the function *create_text_files*. Using the output from *build_bitmaps_and_targets*, create a text file for each bitmap.
    a. Each file should contain 33 lines:
        i. The first line contains a single character – the digit that this sample represents (0 – 9).
        ii. The remaining 32 lines are binary strings, each containing 32 1s and 0s.
    b. The files should be named according to this pattern:

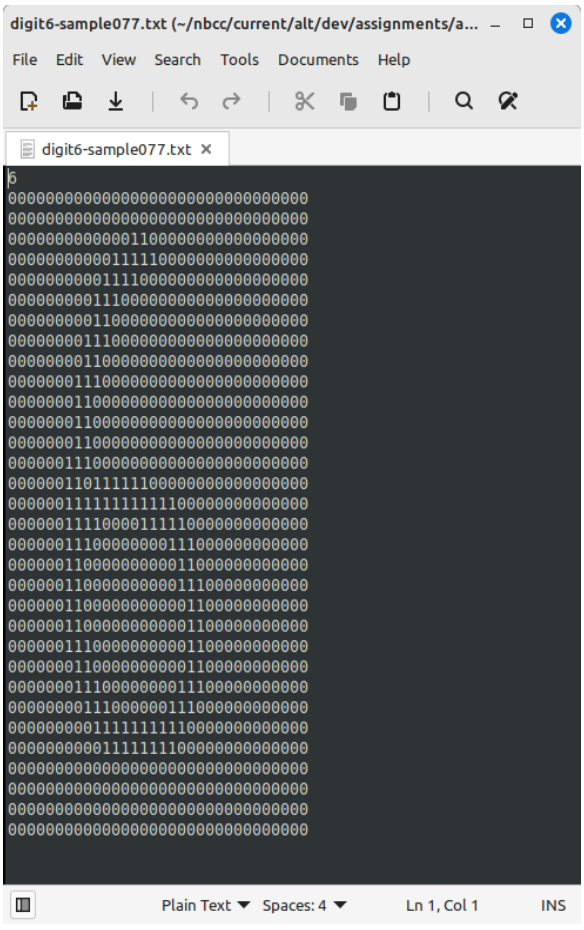
        `digitX-sampleYYY.txt`

        where X is the same as the digit on the first line of the file, and YYY ranges from 1 to the number of samples for that digit. (Note that sample numbers less than 100 should be left-padded with 0s – for example "digit6-sample053.txt"

4. Run the program `a6_part1.py`
    a. Verify that the grayscale images have been saved in the "grayscaleSamples" folder.
    b. Verify that the text bitmap files have been saved in the "textSamples" folder.


originalSamples
File Edit View Go Bookmarks Help
alt dev assignments a6 a6solution originalSamples
1,696 items, Free space: 175.0 GB
grayscaleSamples
File Edit View Go Bookmarks Help
alt dev assignments a6 a6solution grayscaleSamples
1,696 items, Free space: 175.0 GB

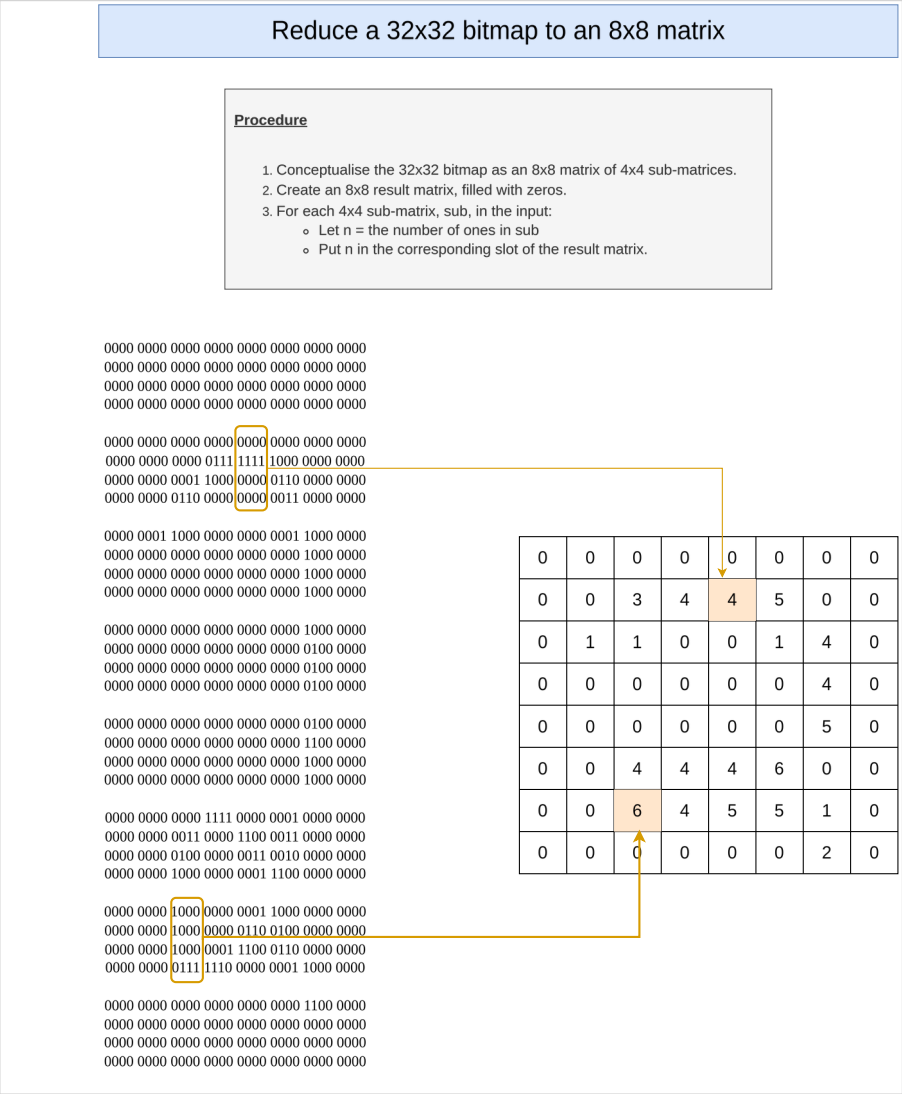

textSamples
File Edit View Go Bookmarks Help
alt dev assignments a6 a6solution textSamples
"digit6-sample014.txt" selected (1.1 kB), Free space: 175.0 GB


digit6-sample077.txt (~/nbcc/current/alt/dev/assignments/a...
File Edit View Search Tools Documents Help
digit6-sample077.txt
6
Plain Text Spaces: 4 Ln 1, Col 1 INS

Part 2 – Reducing Dimensionality and Classifying (file: a6_part2.py)

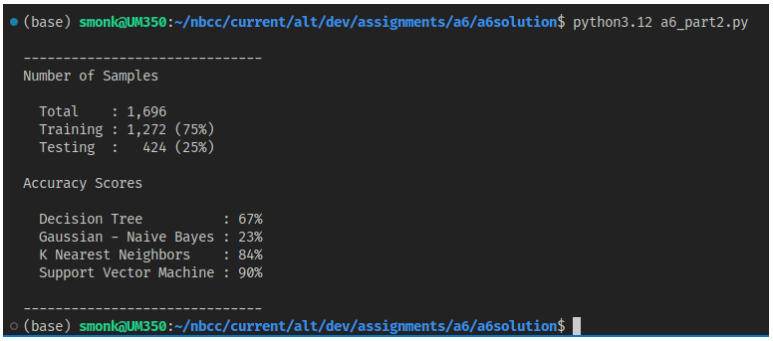
1. Write the function *read_images_and_targets_from_text_files*. This function is similar to *build_bitmaps_and_targets* (from Part 1) in the sense that it returns a list of bitmaps and a list of targets. But this function reads the data from text files, rather than from the images.

2. Write the function *reduce_dimensions*. This function should convert each 32x32 bitmap to an 8x8 matrix. Do this by counting the number of 1's in each non-overlapping 4x4 block.
    - Note that the classification will still work if you skip this step, but the results will be less accurate.
    - If you wish, you could skip this function temporarily and proceed to step 3. Then, when everything else is working, come back and finish this function.



3. Write the function *flatten_images*. This function should convert each two-dimensional image map into a one-dimensional list – i.e., each 8x8 matrix should become a 1x64 list. (Or if you skipped step 2, then each 32x32 matrix would become a 1x1024 list.)
    - This step is required because the *train_test_split* function provided by scikit-learn requires one-dimensional data.

4. Write the function *train_predict_report*. This function should:
    a. Run the four classifiers shown in the sample below using the flattened bitmaps and targets that you have prepared in the preceding steps.
    b. Prepare a report and display it to the console. The report should have two sections:
        i. The number of samples used in the training and testing sets.
        ii. The accuracy scores for each classifier.



    c. Create and save a PNG file for each classifier, showing the overall accuracy score and the confusion matrix. (See next page for samples.)

What to Submit

- The code for parts 1 and 2.
- The original samples used, placed in the original samples folder.
- A screen shot of the report produced in part 2.
- The four PNG files produced in part 2.
- Make sure that I can run part 1 to recreate the grayscale and text files and run part 2 to produce the report and create the PNG files.

Sample PNG Files