Easy Handwriting Recognition

Eric D. Manley, Timothy Urness
Department of Mathematics and Computer Science
Drake University
Des Moines, IA 50311
{eric.manley,timothy.urness}@drake.edu

Abstract

In this assignment, students create a GUI which allows the user to draw handwritten digits using a mouse and then attempts to recognize the character using a CS1-accessible machine learning algorithm. The assignment can be used to emphasize file input, 2D lists/arrays, and/or GUIs. The assignment is nifty because it uses real data with a simple algorithm to achieve compelling results for a familiar application.

1 The GUI

The students create a GUI consisting of a canvas which responds to click-move mouse events by changing the color of the canvas pixel and capturing the corresponding entry in a 2D-List of 0s and 1s, a button which runs the prediction algorithm, and a label to display the algorithm's guess. The example GUI in Figure 1 was made using the Python *tkinter* module, though any GUI with a similar canvas (and any programming language) will work.

2 The Data

The user's drawing is compared against a set of handwritten digit samples from the well-known MNIST database [1], which have been reformatted for CS1-accessible reading. We converted the data from grayscale to black and white for easier comparison, reduced the number of samples (from 60000 to 2000), and converted it from binary to a comma separated values (csv) file where every 29 lines consists of a line with the digit represented and then 28 lines representing the rows in the 28x28 pixel image (see Figure 1). The resulting data set has an easy-to-understand format, where the handwritten

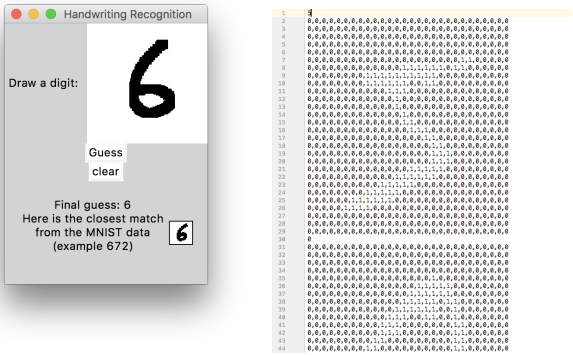

Figure 1: The GUI and Data

characters can be seen visually in the 0s and 1s, and students see immediately how it should be read in as 2D-lists.

3 The Prediction Algorithm

Students write prediction code as follows:

- Create a function computes a similarity score between the drawing 2D list and a sample from the data set. You can simply count the number of pixels that agree or weight some more heavily than others (e.g., 5 points for black pixel agreement, 1 point for white). As an optional extension, students can be given the freedom to experiment and define similarity in any way they choose.
- Loop through each of the samples in the data set, compute the similarity between that sample and the drawing, and keep track of the sample with the best similarity score. The digit corresponding to the best similarity score is then displayed as the prediction.

This algorithm is a variant of the K-nearest-neighbor (KNN) machine learning algorithm with $k = 1$ and a custom similarity function. Other variations of KNN are known to perform very well on this data set [1].

References

[1] Yann LeCun, Corinna Cortes, and Christopher J.C. Burges. The MNIST database. `http://yann.lecun.com/exdb/mnist/`.