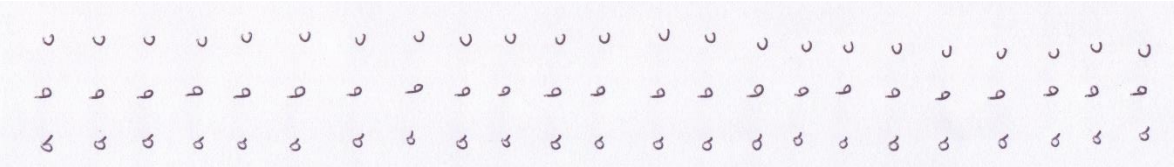
PIC 16, Winter 2018 – Assignment 10W

Assigned 3/14/2018. Code (a `.zip` of your `.py` file and handwriting data image files) due 12 p.m. 3/19/2018 on CCLE. Hand in a printout of this document with the self-assessment portion completed during the final exam on 3/21/2018.

In this assignment, you will train a Support Vector Classifier to read your own handwriting[1].

Task

1. Before you begin, you might want to see the example Recognizing Hand-Written Digits from the scikit-learn documentation. You need to conceptually understand what the `data` and `target` arrays represent before you continue.
   In the example, the `data` and `target` arrays are provided. Training and testing the SVC is trivial once you have these arrays. Therefore, much of the time of your time on this assignment will be spent generating `data` and `target` arrays for your own handwriting.
2. (30pts; 40 pts w/ extra credit) In order to train the SVC, you'll need to provide it with several handwritten examples of each symbol you want it to learn, and you'll need to format this information like the `data` and `target` arrays in the example.
   For example, I wanted my SVC to learn to recognize my handwritten letters "a", "b", and "c" , so I wrote my letters in a grid like:

   manually separated the image into separate files for each letter like:

   and wrote an algorithm to automatically separate the image into an array of images, each tightly cropped around a single symbol like:

   I then converted these into the `data` array required by NumPy. I used the number 0 to represent 'a', 1 to represent 'b', and 2 to represent 'c' in the `target` array.
   I have provided some code (`separate.py`) for converting a column (rotated above to fit on page) of single characters into a list of separate images using a very simple algorithm. You are welcome to use it if you find it helpful, *but for 10 extra points, write your own routine using OpenCV*. (Hint: find contours; calculate bounding rectangles; filter out, based on area, small bounding rectangles that belong to stray marks and other invalid contours; and generate a list of individual letter images by selecting indices from the original image that correspond with the remaining bounding rectangles.)
   Regardless, it is up to you to figure out how to convert this list into the `data` and `target` arrays required by scikit-learn. Do this for *my* handwriting data.
   Tip: Remember that each of the little images provided by `separate` are 2D NumPy arrays of different shapes. You'll need to process these so that each row of `data` represents a single *sample*

[1] Samples of my handwriting are included with the assignment for testing purposes, but you should train your SVC to recognize at least three symbols of *your own* handwriting and include the train/test data with your submission. I didn't know how well this would work, so I trained the SVC with many examples of just a few symbols. Since it worked very well, I wish I had instead trained the machine with just a few examples of many symbols. After you get your code working, I suggest you try that!

with the same number of *features*. I resized each of my letters to 5px square before assembling the `data` array.

3. (20 pts) Generate your own handwriting data: using a dark pen on a white sheet of paper, write in a column several examples of a single symbol. Do this for at least three different symbols. Scan or photograph (cell phone cameras can work) your data and separate it into separate images for each symbol like I did. Generate `data` and `target` arrays for your handwriting data.
   Despite the fact that I provide 23 examples of each letter, five examples of each symbol should be sufficient if your handwriting is consistent. I suggest writing out the whole alphabet five times if you think having a program that recognizes your handwriting would be cool.
   If you really want to have some fun, write out both lower and uppercase letters. I haven't done this, but I imagine there will be some issues distinguishing between lower- and upper- case versions of some letters, like "o" vs "O". To help with this, you could include some information about the size of the letter (like the number of pixels) as an additional feature.
4. (10 pts) Once you have arrays of all your own handwriting data and target values, you need to separate these into test and training sets. Write a function `partition` that accepts your `data` and `target` arrays and a third parameter `p` representing the percentage of the data that is to be used for training; the remainder will be for testing. It should return `train_data`, `train_target`, `test_data`, and `test_target`. Remember that if your samples are ordered, selecting the *first N* samples isn't appropriate; you need to select an assortment from each.
5. (20 pts) Train and test a `LinearSVC` rather than a regular `SVC`. For this problem, `LinearSVC` performance seems much better, and no parameters are needed in the constructor! Print results like:
   ```
   Predicted:  [ 2.  0.  1.  1.  2.  2.]
   Truth:  [ 2.  0.  1.  1.  2.  2.]
   Accuracy:  100.0 %
   ```
6. (10 pts) Was it lucky that your SVC worked as well as it did? Would it behave the same way even if:
   - different samples had been used for training and testing?
   - the number of training samples for each class were not exactly equal?
   - the order in which the training data samples were provided had been different?

   If necessary, modify your partition function so that your training and test information varies to check whether your SVC's performance depends on these factors. In the end, my LinearSVC didn't depend at all on these factors. In fact, it tended to distinguish among the three characters perfectly as long as it was trained on at least one sample of each!
7. (10 pts) Now train and test a (nonlinear) `SVC`. Does it work with the value of `gamma` used in the example? If not, try other values of `gamma`.
   My nonlinear `SVC` didn't work very well unless I used a very low value of `gamma` or specified `kernel = "poly"` manually. Sometimes nonlinear SVCs can be better, but in this case, a linear SVC is all we need.
8. (Optional) Write any additional code needed to read a line of text, and test your SVC on a handwritten sentence. (Show me for golden ticket.)

Self-Assessment

Print the assignment document and check off the steps you completed successfully. Assign points as indicated for each part and write your total score below.