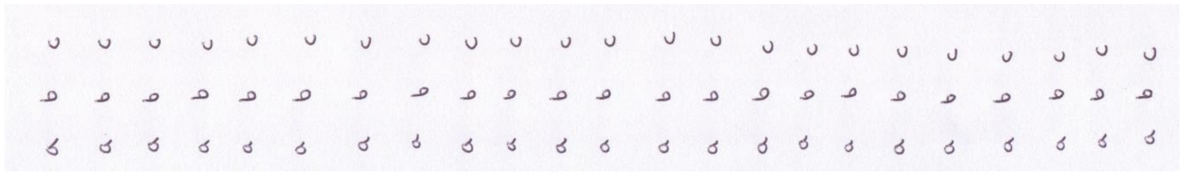
PIC 97, Winter 2016 – Assignment 10M

Assigned 3/7/2016. Code (a `.zip` of your `.py` file and image files) due 12p.m. 3/9/2016 on CCLE. Hand in a printout of this document with the self-assessment portion completed by the end of class on 3/9/2016.

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 review 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.
   Here, the `data` and `target` arrays are provided. Training and testing the SVC is trivial once you have these arrays. Therefore, the majority of your time on this assignment will be spent generating `data` and `target` arrays for your own handwriting.
2. 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 drew 23 instances of each of these letters on a sheet of paper; scanned the sheet; processed the image to generate an array of small images, each cropped around a single symbol; and from this generated the `data` and `target` arrays required by NumPy.
   More specifically, 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 the code for converting a column (rotated above to fit on page) of single characters into a list of separate images. You are welcome to use it if you find it helpful, but it would be much cooler if you wrote your own routine using OpenCV. Regardless, it is up to you to figure out how to convert this into the `data` and `target` arrays. Remember that each of the little images provided by my code are 2D NumPy arrays of different shapes; you'll need to process these so that each row of `data` represents a single *sample* with the same number of *features*. Hint: I resized each of my letters to 5px square before assembling the `data` array.

[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!

3. Once you have arrays of all your 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`.
4. 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 out the results like:
```
Predicted:  [ 2.  0.  1.  1.  2.  2.]
Truth:  [ 2.  0.  1.  1.  2.  2.]
Accuracy:  100.0 %
```
5. 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`!
6. Was it lucky that your SVCs worked as well as they did? Would they 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?

   Modify your `partition` function (as necessary) so that your training and test information varies to test the dependence of your SCV's performance on these factors. Note that the third argument `p` is now an *approximate* percentage of data to be used for training.
   In the end, my `LinearSVC` didn't depend at all on these factors. It tended to distinguish among these characters perfectly as long as it was trained on at least one sample of each! My nonlinear `SVC`, on the other hand, depended quite heavily on these factors for low values of `gamma`. Sometimes nonlinear SVCs can be better, but in this case, a linear SVC is all we need.

7. (Optional) Write any additional code needed to read a line of text, and test your SVC on a handwritten sentence.

Self-Assessment

Print the assignment document and check off the steps you completed successfully.