University of San Francisco

Physics 301 – Computational Physics
Fall, 2016

Final Project: *Face Detection and Recognition*

Due at 11 PM on Monday, December 12, 2016

There are two parts to this project. They share the same library of python functions, which is described below. Part I takes a more in-depth look at one subtle aspect of pattern recognition. Part II is a "proof of concept" exercise of face detection and recognition.

In what follows,

- I will refer to the data array as `X`, and `y` as its target (or ground truth). For example, for Part I below, `X` refers to the sklearn digit image data set.
- `md_pca` is an object of the class `sklearn.decomposition.PCA`.
- `md_clf` is an object of the class `sklearn.svm.SVC`.

Library of Python Functions

Save all the functions needed in parts I and II in one file:

`pattern_recog_func.py`

and they include the following.

1. `interpol_im(im, dim1 = 8, dim2 = 8, plot_new_im = False,\`
`cmap = 'binary', axis_off = False)`

   It should interpolate the input image, `im`, onto a grid that is `dim1` × `dim2`, referring to the number of pixels for the horizontal and vertical directions, respectively. If `plot_new_im` is `True`, display the interpolated image using `plt.imshow()`, with the keyword argument `interpolation` set to `"nearest"`. This way, the image will show each pixel as is, without interpolation, making clear useful diagnostic information, such as the number of pixels in the image and which pixels have high (or low) values. Also, as the default, it should show a grid. But if the flag `grid_off` is `True` then turn off `plt.grid`. The function should return the interpolated, flattened image array.

2. `pca_svm_pred(imfile, md_pca, md_clf, dim1 = 45, dim2 = 60)`
   This function will be used in Part II and it
   - loads the image contained in `imfile`;
   - interpolates the image by calling `interpol_im()`;
   - projects the interpolated, flattened image array onto the PCA space by using `md_pca`;
   - makes a prediction as to the identity of the image, by using `md_clf`; and finally,
   - returns the prediction.

3. `pca_X(X, n_comp = 10)`. It returns `md_pca` and `X_proj`, where `X_proj` contains the projections of the data array `X` in the PCA space. When you instantiate the PCA object, remember to set `whiten = True`.

4. `rescale_pixel(X, unseen, ind = 0)`

   This function will be used in Part I. It rescales the pixel values of the image `unseen`, so that this image has the same pixel value range (between 0 and 15, or a 4-bit integer) as the images in the sklearn digit data set.

5. `svm_train(X, y, gamma = 0.001, C = 100)`

   It returns the `md_clf`. The meanings of `gamma` and `C` were explained in class. For our purposes, you may leave `gamma` and `C` at their default values for both Parts I and II.

You will write one python program for each of the two parts below. Each should import relevant functions from `pattern_recog_func.py`.

I. The Power of SVM and the Importance of Pixel Scale

Write a program and call it

`dig_recog.py`

that does the following. But first note: ***<u>*There is no PCA involved in this part!!*</u>***

a) Training and Validation. It should use the first 60 images in `X` to train, by calling `svm_train()`, which, as stated above returns `md_clf`. Then apply `md_clf` to the next 20 images to perform validation. It then prints out the success rate, depending on how many of the 20 images in the validation set are correctly identified. Make sure that there is no overlap between the training set and the validation set. In the validation process, print out any image for which the prediction is incorrect. Also print out the total number of mis-identifications and the success rate, like so:

```
--------> index, actual digit, svm_prediction: 69 9 8
Total number of mid-identifications: 1
Success rate: 0.95
```

b) Testing. Test the classifier obtained in part a) on an image taken outside of the sklearn digit data set. I have provided the image file,

`unseen_dig.png`

First load the image, call it

`unseen`

and then interpolate this image so that its dimension is 8 × 8, by using `interpol_im()` with `plot_new_im = True`. This should display the image, with `plt.imshow()`, for human inspection — note that it's a "5" and it's a little smaller than the "5" in `X[15]` — you should display this "5" as well. Then by calling `rescale_pixel()`, rescale the pixel values of the interpolated `unseen` so that

i) it has a pixel value range between 0 and 15. Note that this means the pixel values are zero for the background;
ii) all the pixel values are integers,

As the last step of this part, it uses the classifier obtained in part a), `md_clf`, to make a prediction for the interpolated `unseen` — both the rescaled version and the un-rescaled version.

You will see that the prediction for the rescaled version is correct, but that for the un-rescaled version is not.

Additional notes:

- When you manipulate an image sometimes it's better to make a copy of it first so that the original image is not tampered with (`import copy`).

- When you apply `md_pca` or `md_clf` to a single image (or its PCA projection), you need to apply the method `.reshape(1, -1)` to the image.

  To understand what `.reshape(1, -1)` does, in your notebook try the following three lines of code

```
>>> a = np.arange(10)
>>> b = a.reshape(1, -1)
```

```
>>> print(a.shape, b.shape)
```

and you should get

```
(10, ) (1, 10)
```

- You may now be asking the question: What do we need PCA for? If you are dealing with a large data set, the reduction of dimensionality makes the computation much more efficient.

- One of the objective of Part I is to show you that that an import aspect of the pattern recognition problem (digits, characters, faces) is image preparation — which includes cropping, interpolation, and possibly, rescaling as well as rounding pixel values. (The digit images provided by the sklearn, e.g., have been prepared in a uniform way.) Otherwise even a sophisticated classifier such as SVM can be thrown off and give erroneous predictions.

II. Face Detection and Recognition with openCV, PCA, and SVM

This is a simple face detection and recognition problem. Write a program and call it

`face_detect_recog.py`

a) Face Detection using openCV — I will show you the steps of installation and how to use it to pick out the faces of three people in a picture.

b) Data set preparation. You are given 120 images, 40 each for same three people. You can crop the images very closely by using openCV so that our classifier will less likely be confused by background features. Interpolate them onto a grid of 45 × 60, referring to the number of pixels for the horizontal and vertical directions respectively, by using

`interpol_im()`.

Then use `numpy.vstack()` to create a data array that contains all the flattened images — let's call this array `X` — and has the shape (90, 2700). Finally construct a list that is the target for these images, `y`. It should have a length of 90, and should have three kinds of elements, 0, 1, and 2, corresponding to the keys of the following dictionary, which you should include in your program:

```
names_dict = {0: 'name0', 1: 'name1', 2: 'name2'}
```

c) Training and Validation. First perform PCA on these images, by using

`pca_X(X, n_comp = 50)`,

which, as stated earlier, returns `md_pca` and `X_proj`. Then do the training on `X_proj` by using

`svm_train()`,

which, also as stated earlier, returns `md_clf`.

The validation should be done by using the leave-one-out method we talked about in class — in each iteration, first apply the same `md_pca` on that singled out image, then apply `md_clf`, and see if the prediction agrees with the corresponding target. Finally, print the success rate.

In general, to distinguish thousands (or more, many more) faces, one often needs over 100 PCA components. But in the case of just three persons, 50 components give a respectable success rate (as far as facial recognition goes).

c) Testing. You will now use your trained SVM classifier to classify the three faces detected in the picture in part a).

Just as in part a), interpolate (by calling `interpol_im()`), and project the interpolated images onto the PCA space by using the `md_pca` obtained in part a). Then make a prediction for these three images by using the `md_clf` obtained in part a). All this should be done in the function

`pca_svm_pred()`

Then by using

`names_dict`

print out the following two statements:

```
PCA+SVM predition for person 0: name0
PCA+SVM predition for person 1: name1
PCA+SVM predition for person 2: name2
```

Don't forget to display the three images for human inspection and verification, with `plt.show()`, with `grid_off = True`. You may do this when you call `interpol_im()`.

Finally print the names of the three faces that have been detected and recognized by your algorithm on the original picture in part a).