University of San Francisco

Physics 301 – Introduction to Scientific Computation
Spring, 2020

Final Project: *Face Detection and Recognition*

Due at 11 PM on Thursday, May 14, 2020

In what follows,

- I will refer to the data array as `X`, and `y` as its target (or ground truth). For this project, `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 for this project in one file:

`pattern_recog_func.py`

They include the following.

1. `interpol_im(im, dim1=8, dim2=8, plot_new_im=False,\`
`cmap='binary', grid_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 useful diagnostic information clear, 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
   - 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(unseen)`

   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.

   The image may also need to be inverted (so that it appears the same way as the images in the training data set), and please make sure the minimum pixel values are 0, as emphasized below.

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

   It returns the `md_clf`. For our purposes, you may leave `gamma` and `C` at their default values. (If you want to know more about the `C` parameter, see https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html.)

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

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: ****There is no PCA involved for parts a) and b)!!****

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. You will see 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.

c) Train the `SVM` classifier in PCA space, by using `X_proj` instead of `X`. To classify `unseen`, after rescaling and interpolation (same as part b)), find its projection in PCA space, and then make the prediction. All this should be done in the function `pca_svm_pred()`.

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 as shown in this case we can successfully classify without using PCA? The answer is that if you are dealing with a large data set, the reduction of dimensionality makes the computation much more efficient.

- One of the objectives of this project is to show you that an important aspect of the pattern recognition problem (digits, characters, faces) is image preparation ("preprocessing") — 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.