Recognizing Handwritten Digits

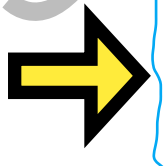
In the previous Internship Project you have seen how to apply the techniques of data analysis to Pandas data-frames containing numbers and strings. Indeed, data analysis is not limited to numbers and strings, because images and sounds can also be analyzed and classified.

Handwriting Recognition

Recognizing handwritten text is a problem that can be traced back to the first automatic machines that needed to recognize individual characters in handwritten documents. Think about, for example, the ZIP codes on letters at the post office and the automation needed to recognize these five digits. Perfect recognition of these codes is necessary in order to sort mail automatically and efficiently.
Included among the other applications that may come to mind is OCR (Optical Character Recognition) software. OCR software must read handwritten text, or pages of printed books, for general electronic documents in which each character is well defined.

But the problem of handwriting recognition goes farther back in time, more precisely to the early 20th Century (1920s), when Emanuel Goldberg (1881–1970) began his studies regarding this issue and suggested that a statistical approach would be an optimal choice.

To address this issue in Python, the **`scikit-learn` library** provides a good example to better understand this technique, the issues involved, and the possibility of making predictions.



This Colab Notebook from Rocky Sir's Data Science would help you to learn Support Vector Machine (or SVC Supervised ML Algo).
https://drive.google.com/open?id=1hr-MZHyTiVRl7Jlfh51DwaO3hQgc80oe

In this Internship doc, SVC is used. Of course, you are free to use any other ML Classifier.

Recognizing Handwritten Digits with scikit-learn

The `scikit-learn` library (`http://scikit-learn.org/`) enables you to approach this type of data analysis in a way that is slightly different from what you've used in **Project 1**. The data to be analyzed is closely related to numerical values or strings, but can also involve images and sounds.

The problem you have to face in *this Internship project* involves predicting a numeric value, and then reading and interpreting an image that uses a handwritten font.

So even in this case you will have an *estimator* with the task of learning through a `fit()` function, and once it has reached a degree of predictive capability (a model sufficiently valid), it will produce a prediction with the `predict()` function. Then we will discuss the training set and validation set, created this time from a series of images.

Now open a new IPython Notebook session from the command line by entering the following command:

```
ipython notebook
```

Then create a new Notebook by choosing New ➤ Python 3, as shown in Figure 1.

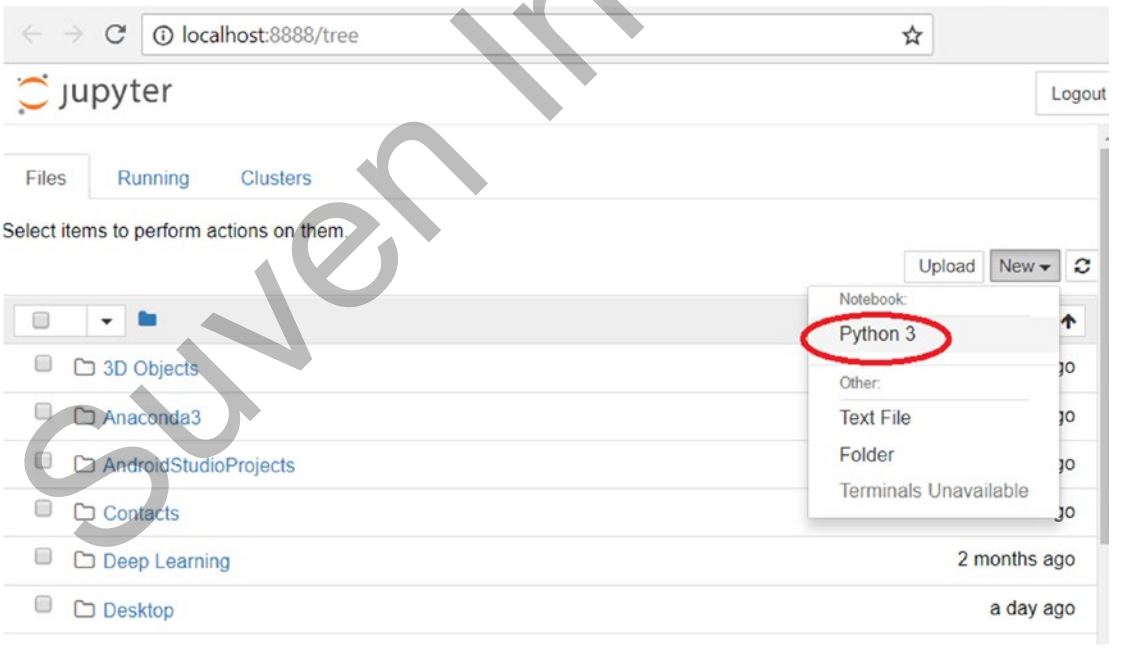


Figure 1. *The home page of the IPython Notebook (Jupyter)*

An estimator that is useful in this case is **sklearn.svm.SVC**, which uses the technique of **Support Vector Classification (SVC)**.

Thus, you have to import the **svm** module of the **scikit-learn** library. You can create an estimator of SVC type and then choose an initial setting, assigning the values `C` and `gamma` generic values. These values can then be adjusted in a different way during the course of the analysis.

```
from sklearn import svm
svc = svm.SVC(gamma=0.001, C=100.)
```

The Digits Dataset

The `scikit-learn` library provides numerous datasets that are useful for testing many problems of data analysis and prediction of the results. Also in this case there is a dataset of images called *Digits*.

This dataset consists of **1,797 images that are 8x8 pixels** in size. Each image is a handwritten digit in grayscale, as shown in Figure 2.

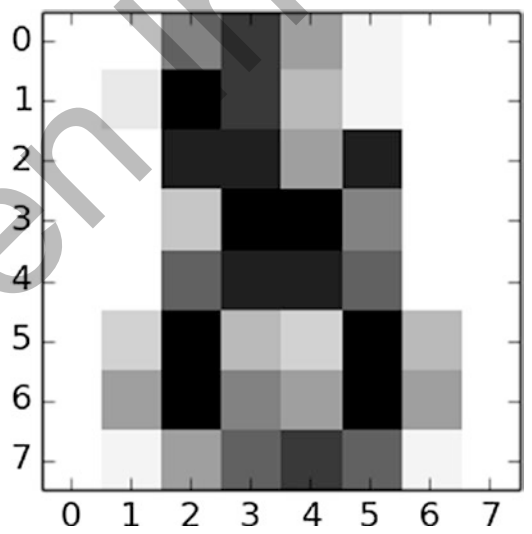


Figure 2. *One of 1,797 handwritten number images that make up the dataset digit*

Thus, you can load the Digits dataset into your Notebook.

```
from sklearn import datasets
digits = datasets.load_digits()
```

After loading the dataset, you can analyze the content. First, you can read lots of information about the datasets by calling the DESCR attribute.

```
print(digits.DESCR)
```

For a textual description of the dataset, the authors who contributed to its creation and the references will appear as shown in Figure 3.

```
print digits.DESCR
 Optical Recognition of Handwritten Digits Data Set

Notes
-----
Data Set Characteristics:
    :Number of Instances: 5620
    :Number of Attributes: 64
    :Attribute Information: 8x8 image of integer pixels in the range 0..16.
    :Missing Attribute Values: None
    :Creator: E. Alpaydin (alpaydin '@' boun.edu.tr)
    :Date: July; 1998

This is a copy of the test set of the UCI ML hand-written digits datasets
http://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits

The data set contains images of hand-written digits: 10 classes where
each class refers to a digit.

Preprocessing programs made available by NIST were used to extract
normalized bitmaps of handwritten digits from a preprinted form. From a
total of 43 people, 30 contributed to the training set and different 13
to the test set. 32x32 bitmaps are divided into nonoverlapping blocks of
4x4 and the number of on pixels are counted in each block. This generates
an input matrix of 8x8 where each element is an integer in the range
0..16. This reduces dimensionality and gives invariance to small
distortions.

For info on NIST preprocessing routines, see M. D. Garris, J. L. Blue, G.
T. Candela, D. L. Dimmick, J. Geist, P. J. Grother, S. A. Janet, and C.
L. Wilson, NIST Form-Based Handprint Recognition System, NISTIR 5469,
1994.
```

Figure 3. *Each dataset in the scikit-learn library has a field containing all the information*

The images of the handwritten digits are contained in a **digits.images** array. Each element of this array is an image that is represented by an 8x8 matrix of numerical values that correspond to a grayscale from white, with a value of 0, to black, with the value 15.

```
digits.images[0]
```

You will get the following result:

```
array([[  0.,   0.,   5.,  13.,   9.,   1.,   0.,   0.],
       [  0.,   0.,  13.,  15.,  10.,  15.,   5.,   0.],
       [  0.,   3.,  15.,   2.,   0.,  11.,   8.,   0.],
       [  0.,   4.,  12.,   0.,   0.,   8.,   8.,   0.],
       [  0.,   5.,   8.,   0.,   0.,   9.,   8.,   0.],
       [  0.,   4.,  11.,   0.,   1.,  12.,   7.,   0.],
       [  0.,   2.,  14.,   5.,  10.,  12.,   0.,   0.],
       [  0.,   0.,   6.,  13.,  10.,   0.,   0.,   0.]])
```

You can visually check the contents of this result using the matplotlib library.

```
import matplotlib.pyplot as plt
%matplotlib inline
plt.imshow(digits.images[0], cmap=plt.cm.gray_r, interpolation='nearest')
```

By launching this command, you will obtain the grayscale image shown in Figure 4.

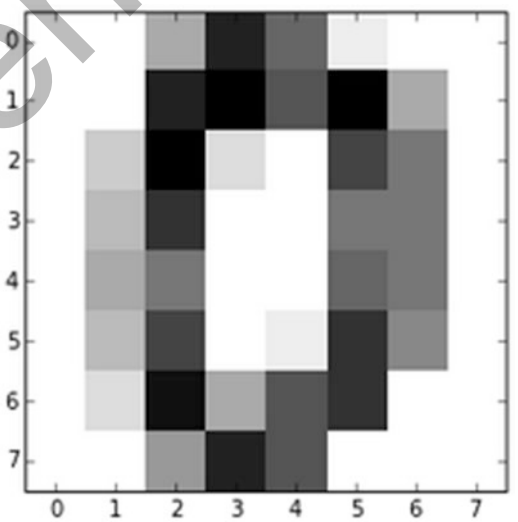


Figure 4. *One of the 1,797 handwritten digits*

The numerical values represented by images, i.e., the targets, are contained in the **digit.targets** array.

```
digits.target
```

You will get the following result:

```
array([0, 1, 2, ..., 8, 9, 8])
```

It was reported that the dataset is a training set consisting of 1,797 images. You can determine if that is true.

```
digits.target.size
```

This will be the result:

```
1797
```

Recall the Hypothesis to be tested : The Digits data set of scikit-learn library provides numerous data-sets that are useful for testing many problems of data analysis and prediction of the results. Some Scientist claims that it predicts the digit accurately 95% of the times. Perform data Analysis to accept or reject this Hypothesis.

Learning and Predicting

Now that you have loaded the Digits datasets into your notebook and have defined an SVC estimator, you can start learning.

You should be knowing that, once you define a predictive model, you must instruct it with a training set, which is a set of data in which you already know the belonging class. Given the large quantity of elements contained in the Digits dataset, you will certainly obtain a very effective model, i.e., one that's capable of recognizing with good certainty the handwritten number.

This dataset contains 1,797 elements, and so you can consider the first 1,791 as a training set and will use the **last six** as a validation set.

You can see in detail these six handwritten digits by using the matplotlib library:

```
import matplotlib.pyplot as plt
%matplotlib inline

plt.subplot(321)
plt.imshow(digits.images[1791], cmap=plt.cm.gray_r,
interpolation='nearest')
plt.subplot(322)
plt.imshow(digits.images[1792], cmap=plt.cm.gray_r,
interpolation='nearest')
```

```
plt.subplot(323)
plt.imshow(digits.images[1793], cmap=plt.cm.gray_r,
interpolation='nearest')
plt.subplot(324)
plt.imshow(digits.images[1794], cmap=plt.cm.gray_r,
interpolation='nearest')
plt.subplot(325)
plt.imshow(digits.images[1795], cmap=plt.cm.gray_r,
interpolation='nearest')
plt.subplot(326)
plt.imshow(digits.images[1796], cmap=plt.cm.gray_r,
interpolation='nearest')
```

This will produce an image with six digits, as shown in Figure 5.

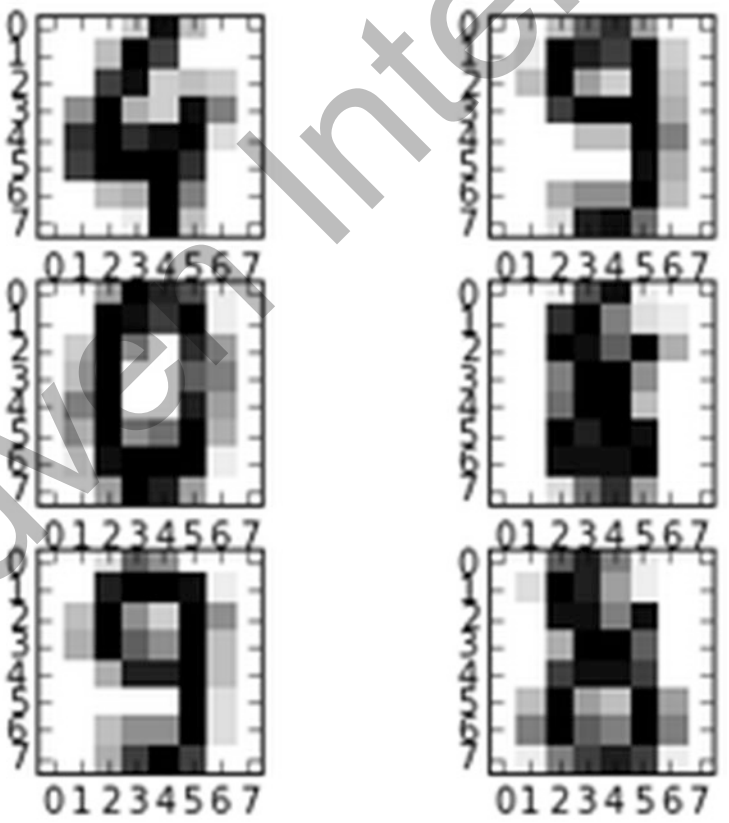


Figure 5. *The six digits of the validation set*

Now you can train the svc estimator that you defined earlier.

```
svc.fit(digits.data[1:1790], digits.target[1:1790])
```

After a short time, the trained estimator will appear with text output.

```
SVC(C=100.0, cache_size=200, class_weight=None, coef0=0.0, degree=3,
  gamma=0.001, kernel='rbf', max_iter=-1, probability=False,
  random_state=None, shrinking=True, tol=0.001, verbose=False)
```

Now you have to test your estimator, making it interpret the six digits of the validation set.

```
svc.predict(digits.data[1791:1796])
```

You will obtain these results:

```
array([4, 9, 0, 8, 9, 8])
```

If you compare them with the actual digits, as follows:

```
digits.target[1791:1796]
```

```
array([4, 9, 0, 8, 9, 8])
```

You can see that the svc estimator has learned correctly. It is able to recognize the handwritten digits, interpreting correctly all six digits of the validation set.

You can choose a smaller training set and different range for validation. In the above case we have got 100% accurate predictions, but this may not be the case at all times.

Run for at-least 3 cases , each case for different range of training and validation sets.

Note : You can choose a different dataset than Digits too.