# Recognizing Handwritten Digits with Scikit-Learn

**Recognizing handwritten text** is a problem that traces 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 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 (the 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.

scikit-learn library:

The scikit-learn library (<http://scikit-learn.org/>) enables us to approach this type of data analysis in a way that differs slightly from what we’ve used in the [previous project](https://gayathri1462.medium.com/performing-analysis-of-meteorological-data-9db091151261). I closely related the data to be analyzed to numerical values or strings, but can also involve images and sounds.

Aim:

The primary aim of this project involves predicting a numeric value, and then reading and interpreting an image that uses a handwritten font.

we 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.

The hypothesis to be tested:

The Digits data set of the scikit-learn library provides numerous datasets 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.

The Digits Dataset

The scikit-learn library provides many 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 comprises 1,797 images that are 8x8 pixels in size. Each image is a handwritten digit in grayscale.

Implementation:

Import Dataset

The scikit-learn library has a package of datasets. These datasets are useful for getting a handle on a machine-learning algorithm or library feature.

Import datasets module from sklearn library and load the digits dataset using the **load\_digits()** function.

from sklearn import datasets  
digits = datasets.load\_digits()

Dataset Description

After loading the dataset, we can read the information about the dataset by calling the **DESCR** attribute.

The textual description of the dataset, the authors who contributed to its creation, and the references will appear as shown in the output.

print(digits.DESCR)*Output:*.. \_digits\_dataset:  
  
Optical recognition of handwritten digits dataset  
--------------------------------------------------  
  
\*\*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  
<https://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.  
  
.. topic:: References  
  
 - C. Kaynak (1995) Methods of Combining Multiple Classifiers and Their  
 Applications to Handwritten Digit Recognition, MSc Thesis, Institute of  
 Graduate Studies in Science and Engineering, Bogazici University.  
 - E. Alpaydin, C. Kaynak (1998) Cascading Classifiers, Kybernetika.  
 - Ken Tang and Ponnuthurai N. Suganthan and Xi Yao and A. Kai Qin.  
 Linear dimensionalityreduction using relevance weighted LDA. School of  
 Electrical and Electronic Engineering Nanyang Technological University.  
 2005.  
 - Claudio Gentile. A New Approximate Maximal Margin Classification  
 Algorithm. NIPS. 2000.

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

Targets

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

digits.target*Output:*array([0, 1, 2, ..., 8, 9, 8])

Dataset shape

**Dimensions**of the dataset can be obtained using **data.shape()** function.

digits.data.shape*Output:*(1797, 64)

The output shows that the dataset has 1797 images of 8x8 size.

Images of the handwritten digits are contained in an array

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

digits.images[0]*Output:*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.]])

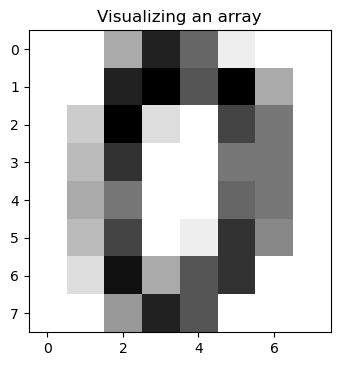
**Visualization of an array**

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

1. Import **pyplot**module which is under matplotlib as **plt**.
2. The **imshow**() function is used to display data as an image; i.e. on a 2D regular raster.
3. **cmap = gray\_r**displays a grayscale image.
4. **interpolation= ‘nearest’** displays an image without trying to interpolate between pixels if the display resolution is not the same as the image resolution.
5. The **title()**function is used to display the title on the graph.

import matplotlib.pyplot as plt  
plt.imshow(digits.images[0], cmap=plt.cm.gray\_r,  
interpolation='nearest')  
plt.title('Visualizing an array')

By running this command, we will obtain the grayscale image as follows

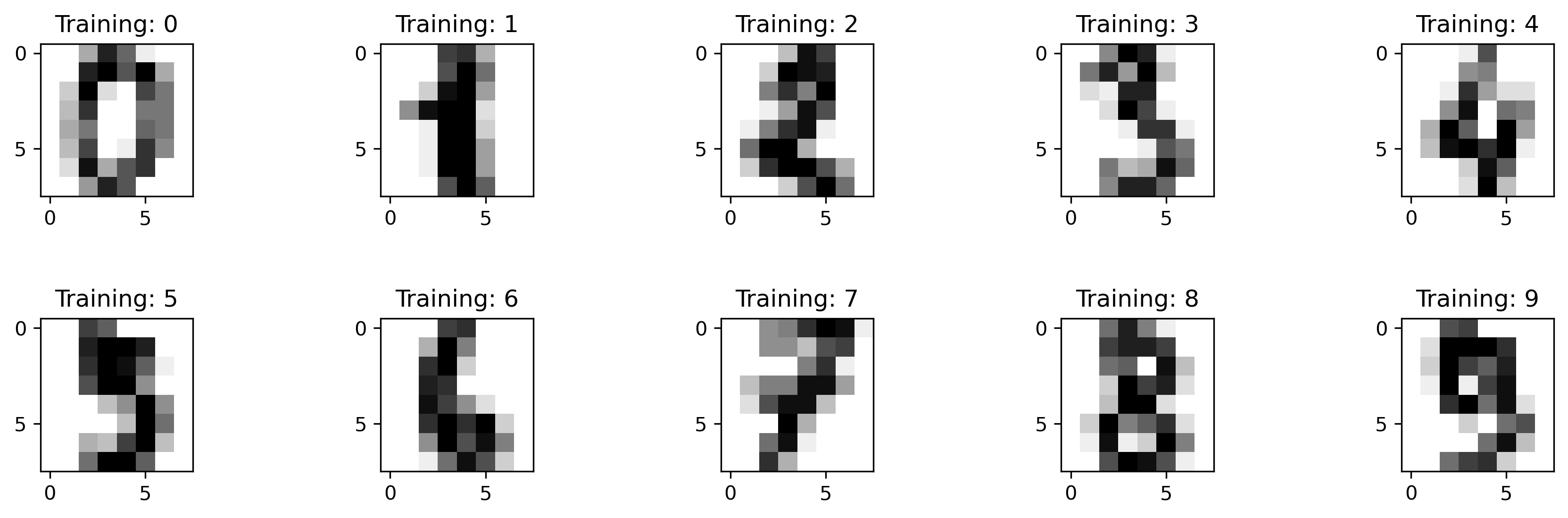


One of the 1797 handwritten digits

**Visualization of the 10 digits**

Using the NumPy and matplotlib libraries, we can display each digit from 0 to 9 which are in the form of an array as images.

1. The **figure()** function in the pyplot module of the matplotlib library is used to create a new **figure**with a specified size of (15,4).
2. **subplots\_adjust(hspace=0.8)**is used to adjust the space between the rows of the subplots.
3. Combine two lists using the **zip()** function for easier handling inside the plotting loop.
4. **enumerate()** method adds a counter to an iterable and returns it. The returned object is a **enumerate** object.
5. **subplot()** function is used to add a subplot to a current figure at the specified grid position.



Training image of each digit

Flatten the input images

The inputs are 8x8 grayscale images. we can produce a flat array of 64-pixel values so that each pixel corresponds to a column for the classifier.

1. **len()** function gives the number of images in the dataset.
2. **reshape()** function returns an array containing the same data with a new shape.

n = len(digits.images)  
print(n)  
data = digits.images.reshape((n, -1))*Output:*1797

It was reported that the dataset is a training set consisting of 1,797 images. We determined that it is true.

**Define the Model**

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

***“Support Vector Machine” (SVM)*** is a supervised machine learning algorithm that is mostly used in classification problems.

Import the SVM module of the scikit-learn library and create an estimator of SVC type and then choose an initial setting, assigning the values C and gamma generic values.

#Import svm model   
from sklearn import svm   
#Create a svm Classifier  
svc = svm.SVC(gamma=0.001, C=100.)

Split the Dataset

once we define a predictive model, we must instruct it with a training and test set. The **training set**is a set of data in which you already know the belonging class and the **test set** is a secondary data set that is used to test a machine learning program after it has been trained on initial training.

Import **train\_test\_split()** function which is used for splitting data arrays into two subsets i.e., into train and test sets.

Here we have split the data by assigning 0.01 as test size.

from sklearn.model\_selection import train\_test\_splitx\_train, x\_test, y\_train, y\_test = train\_test\_split(data, digits.target, test\_size=0.01, random\_state=0)

Train the model

we can train the svc estimator that we defined earlier usingthe**fit()** function.

svc.fit(x\_train, y\_train)

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

SVC(C=100.0, gamma=0.001)

Test the model

we can test our estimator by making it interpret the digits of the test set using **predict()** function.

y\_pred = svc.predict(x\_test)  
y\_pred*Output:*array([2, 8, 2, 6, 6, 7, 1, 9, 8, 5, 2, 8, 6, 6, 6, 6, 1, 0])

We obtain the results in the form of an array.

**Visualize the test images**

We can plot the images of the predicted digits from the array using the following code.



Test images and their predicted values

It is able to recognize the handwritten digits and interprete all the digits of the validation set correctly.

Accuracy of the model

The accuracy score of the model can be obtained using the **score()** function.

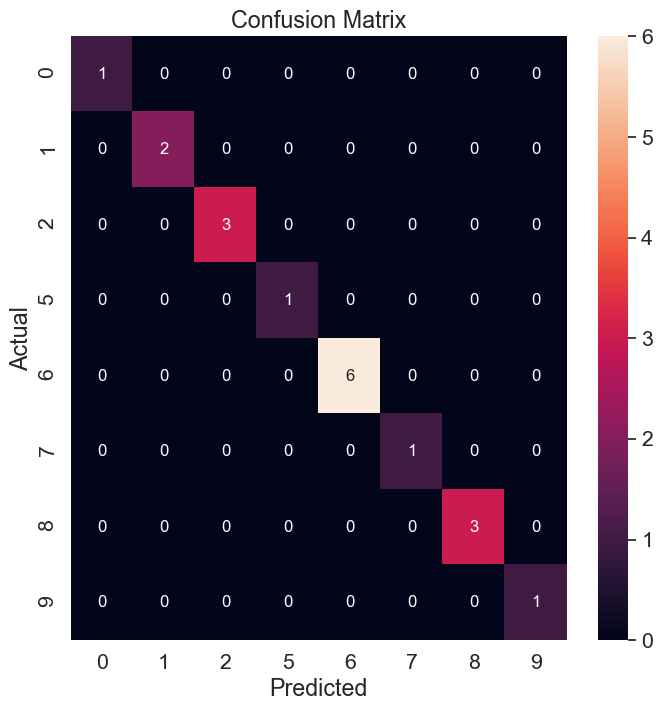
score = svc.score(x\_test, y\_test)*Output:*Accuracy Score: 1.0

Confusion matrix and Classification report of the model

A **confusion matrix** is a table that is often used to describe the performance of a classification model (or “classifier”) on a set of test data for which the true values are known.

A **Classification report** is used to measure the quality of predictions from a **classification** algorithm.

The following code will display the confusion matrix and classification report using the **confusion\_matrix()** and **classification\_report()** functions.

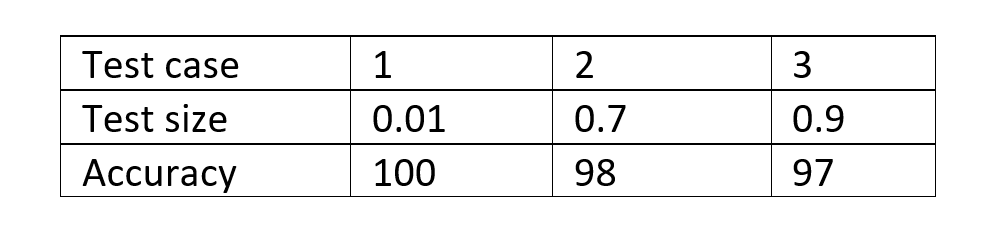


Classification report for SVM classifier:  
  
 precision recall f1-score support  
  
 0 1.00 1.00 1.00 1  
 1 1.00 1.00 1.00 2  
 2 1.00 1.00 1.00 3  
 5 1.00 1.00 1.00 1  
 6 1.00 1.00 1.00 6  
 7 1.00 1.00 1.00 1  
 8 1.00 1.00 1.00 3  
 9 1.00 1.00 1.00 1  
  
 accuracy 1.00 18  
 macro avg 1.00 1.00 1.00 18  
weighted avg 1.00 1.00 1.00 18

Conclusion

Given the large number of elements contained in the Digits dataset, we will certainly obtain a very effective model, i.e., one that’s capable of recognizing with good certainty.

We test the hypothesis by using these cases, each case for a different range of training and validation sets.



Tested the model with 3 different ranges

After performing the data analysis on the dataset with three different test cases, we can conclude that the given hypothesis is true i.e., the model predicts the digit accurately 95% of the times.