Student Detials

> Title= "Mr"\ Name= "Ali Nawaz"\ email = "nawazktk99@gmail.com"\ whatsapp = "03358043653"

In this Notebook I am are going to Explore Image classifcation using Machine Learning with Python

What is Pixel:

A pixel is a contraction if the term PIcture ELement. Digital images are made up of small squares, just like a tile mosaic on a wall. Though a digital photograph looks smooth and continuous just like a regular photograph, it's actually composed of millions of tiny squares

What exactly is a pixel?

The pixel (a word invented from "picture element") is the basic unit of programmable color on a computer display or in a computer image. Think of it as a logical - rather than a physical - unit. The physical size of a pixel depends on how you've set the resolution for the display screen

Types of Images:

The binary image

The binary image as it name states, contain only two pixel values.

0 and 1.

In our previous tutorial of bits per pixel, we have explained this in detail about the representation of pixel values to their respective colors.

Here 0 refers to black color and 1 refers to white color. It is also known as Monochrome.

Black and white image:

The resulting image that is formed hence consist of only black and white color and thus can also be called as Black and White image.

image

black and white

No gray level One of the interesting this about this binary image that there is no gray level in it. Only two colors that are black and white are found in it.

Format Binary images have a format of PBM (Portable bit map)

2, 3, 4,5, 6 bit color format The images with a color format of 2, 3, 4, 5 and 6 bit are not widely used today. They were used in old times for old TV displays, or monitor displays.

But each of these colors have more then two gray levels, and hence has gray color unlike the binary image.

In a 2 bit 4, in a 3 bit 8, in a 4 bit 16, in a 5 bit 32, in a 6 bit 64 different colors are present.

8 bit color format

8 bit color format is one of the most famous image format. It has 256 different shades of colors in it. It is commonly known as Grayscale image.

The range of the colors in 8 bit vary from 0-255. Where 0 stands for black, and 255 stands for white, and 127 stands for gray color.

This format was used initially by early models of the operating systems UNIX and the early color Macintoshes.

A grayscale image of Einstein is shown below: image

einstein Format The format of these images are PGM (Portable Gray Map).

This format is not supported by default from windows. In order to see gray scale image, you need to have an image viewer or image processing toolbox such as Matlab.

Behind gray scale image: As we have explained it several times in the previous tutorials, that an image is nothing but a two dimensional function, and can be represented by a two dimensional array or matrix. So in the case of the image of Einstein shown above, there would be two dimensional matrix in behind with values ranging between 0 and 255.

But thats not the case with the color images.

16 bit color format

It is a color image format. It has 65,536 different colors in it. It is also known as High color format.

It has been used by Microsoft in their systems that support more then 8 bit color format. Now in this 16 bit format and the next format we are going to discuss which is a 24 bit format are both color format.

The distribution of color in a color image is not as simple as it was in grayscale image.

A 16 bit format is actually divided into three further formats which are Red , Green and Blue. The famous (RGB) format.

It is pictorially represented in the image below. image

16-bit Now the question arises, that how would you distribute 16 into three. If you do it like this,

5 bits for R, 5 bits for G, 5 bits for B

Then there is one bit remains in the end.

So the distribution of 16 bit has been done like this.

5 bits for R, 6 bits for G, 5 bits for B.

The additional bit that was left behind is added into the green bit. Because green is the color which is most soothing to eyes in all of these three colors.

Note this is distribution is not followed by all the systems. Some have introduced an alpha channel in the 16 bit.

Another distribution of 16 bit format is like this: 4 bits for R, 4 bits for G, 4 bits for B, 4 bits for alpha channel.

Or some distribute it like this

5 bits for R, 5 bits for G, 5 bits for B, 1 bits for alpha channel.

24 bit color format

24 bit color format also known as true color format. Like 16 bit color format, in a 24 bit color format, the 24 bits are again distributed in three different formats of Red, Green and Blue. image

24-bit Since 24 is equally divided on 8, so it has been distributed equally between three different color channels.

Their distribution is like this.

8 bits for R, 8 bits for G, 8 bits for B.

Behind a 24 bit image.

Unlike a 8 bit gray scale image, which has one matrix behind it, a 24 bit image has three different matrices of R, G, B. image

what is image

Format It is the most common used format. Its format is PPM (Portable pixMap) which is supported by Linux operating system. The famous windows has its own format for it which is BMP (Bitmap).

RGB (red, green, blue):

RGB image is a three-dimensional byte array that explicitly stores a color value for each pixel. RGB image arrays are made up of width, height, and three channels of color information. Scanned photographs are commonly stored as RGB images

CMYK (Cyan, Magenta, Yellow, and Key) (black):

CMYK represent Cyan, Magenta, Yellow, and Key (black). By mixing the four colors in varying amounts, millions of others shades are produced in the printed material. These links are utilized when printing photos in books and magazines. RGB describes the colors of images that are viewed on the monitor

Recognizing hand-written digits

In [ ]:
```
# import all the lib
import pandas as pd
import seaborn as sns
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn import linear_model
import sklearn.metrics as sm
import matplotlib.pyplot as plt

# Importing Linear Regression model from scikit learn
from sklearn.linear_model import LinearRegression
# Importing metrics for the evaluation of the model
from sklearn.metrics import r2_score,mean_squared_error
```

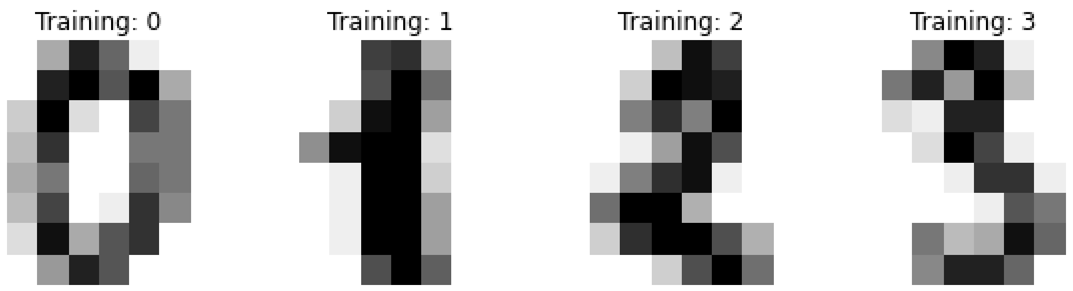
Data Exploration

In [ ]:
```
digits = load_digits()
digits.data.shape
```
```
(1797, 64)
```

In [ ]:
```
digits.target.shape
```
```
(1797,)
```

In [ ]:
```
digits.images.shape
```
```
(1797, 8, 8)
```

In [ ]:
```
_, axes = plt.subplots(nrows=1, ncols=4, figsize=(10, 3))
for ax, image, label in zip(axes, digits.images, digits.target):
    ax.set_axis_off()
    ax.imshow(image, cmap=plt.cm.gray_r, interpolation="nearest")
    ax.set_title("Training: %i" % label)
```



In [ ]:

```
print( digits.images[0])
```

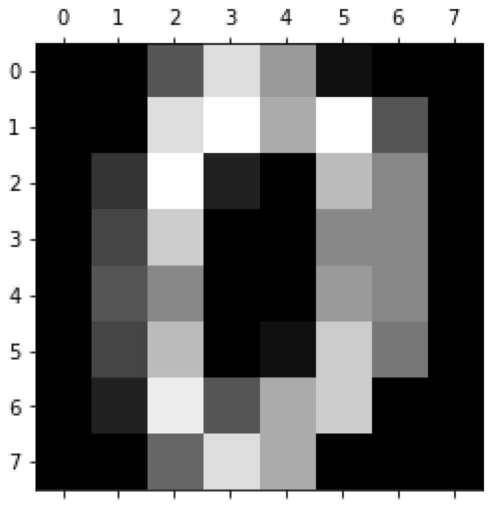
```
[[ 0.  0.  5. 13.  9.  1.  0.  0.]
 [ 0.  0. 13. 15. 10. 15.  5.  0.]
 [ 0.  3. 15.  2.  0. 11.  8.  0.]
 [ 0.  4. 12.  0.  0.  8.  8.  0.]
 [ 0.  5.  8.  0.  0.  9.  8.  0.]
 [ 0.  4. 11.  0.  1. 12.  7.  0.]
 [ 0.  2. 14.  5. 10. 12.  0.  0.]
 [ 0.  0.  6. 13. 10.  0.  0.  0.]]
```

In [ ]:

```
plt.gray()
plt.matshow(digits.images[0])

plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```



In [ ]:

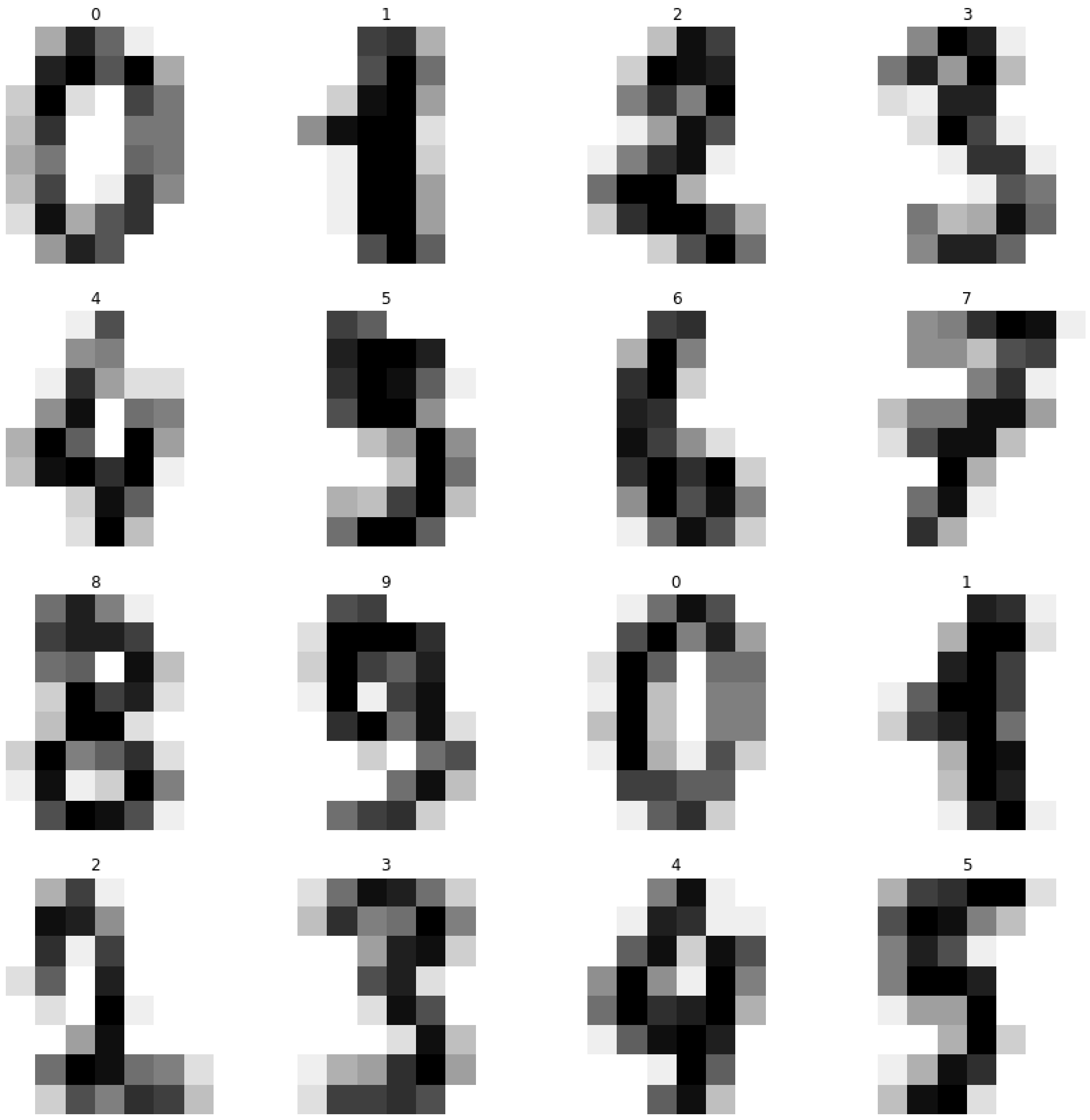
```
digits.target
```

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

In [ ]:

```
def plot_multi(i):
    '''Plots 16 digits, starting with digit i'''
    nplots = 16
    fig = plt.figure(figsize=(15,15))
    for j in range(nplots):
        plt.subplot(4,4,j+1)
        plt.imshow(digits.images[i+j], cmap='binary')
        plt.title(digits.target[i+j])
        plt.axis('off')
    plt.show()
```

In [ ]:
```
plot_multi(0)
```



Building the network and preparing the input data

In [ ]:
```
# Print to show there are 1797 images (8 by 8 images for a dimensionality of 64)
print('Image Data Shape' , digits.data.shape)
# Print to show there are 1797 labels (integers from 0-9)
print("Label Data Shape", digits.target.shape)
```

```
Image Data Shape (1797, 64)
Label Data Shape (1797,)
```

In [ ]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(digits.data, digits.target, test_si
```

In [ ]:
```
print('Input Shape of X Train',X_train.shape)
print('Input Shape of Y Train',y_train.shape)
print('Input Shape of X Test',X_test.shape)
print('Input Shape of Y Test',y_test.shape)
```

```
Input Shape of X Train (1347, 64)
Input Shape of Y Train (1347,)
Input Shape of X Test (450, 64)
Input Shape of Y Test (450,)
```

In [ ]:
```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(X_train, y_train)
```

```
C:\Users\Ali\anaconda3\envs\python-chilla\lib\site-packages\sklearn\linear_model\logisti
c.py:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a so
lver to silence this warning.
  FutureWarning)
C:\Users\Ali\anaconda3\envs\python-chilla\lib\site-packages\sklearn\linear_model\logisti
c.py:460: FutureWarning: Default multi_class will be changed to 'auto' in 0.22. Specify
the multi_class option to silence this warning.
  "this warning.", FutureWarning)
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
```

In [ ]:
```
# Learn the digits on the train subset
model.fit(X_train, y_train)

# Predict the value of the digit on the test subset
predicted = model.predict(X_test)
```

```
C:\Users\Ali\anaconda3\envs\python-chilla\lib\site-packages\sklearn\linear_model\logisti
c.py:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a so
lver to silence this warning.
  FutureWarning)
C:\Users\Ali\anaconda3\envs\python-chilla\lib\site-packages\sklearn\linear_model\logisti
c.py:460: FutureWarning: Default multi_class will be changed to 'auto' in 0.22. Specify
the multi_class option to silence this warning.
  "this warning.", FutureWarning)
```

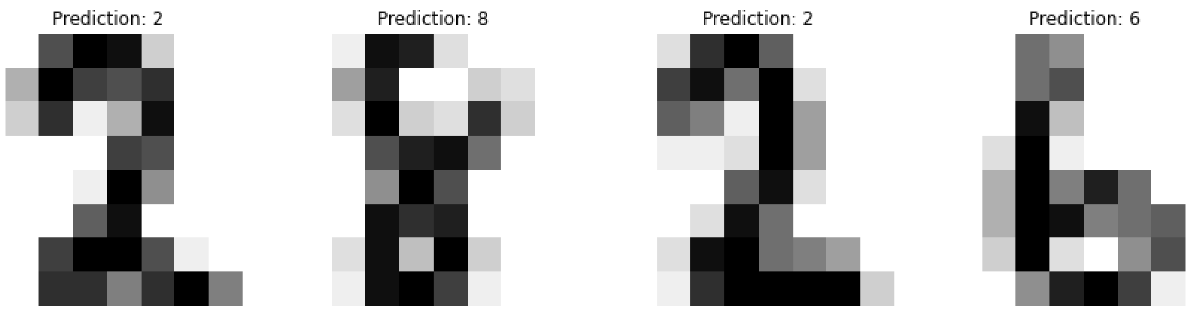
In [ ]:
```
# Returns a NumPy Array
# Predict for One Observation (image)
model.predict(X_test[0].reshape(1,-1))
```

```
array([2])
```

In [ ]:
```
predictions = model.predict(X_test)
score = model.score(X_test, y_test)
# print("Prediction are:", pred)
print("Accuracy Score is :", score)
```

```
Accuracy Score is : 0.9533333333333334
```

In [ ]:
```
_, axes = plt.subplots(nrows=1, ncols=4, figsize=(15, 5))
for ax, image, prediction in zip(axes, X_test, predicted):
    ax.set_axis_off()
    image = image.reshape(8, 8)
    ax.imshow(image, cmap=plt.cm.gray_r, interpolation="nearest")
    ax.set_title(f"Prediction: {prediction}")
```



In [ ]:

```
# Returns a NumPy Array
# Predict for One Observation (image)
model.predict(X_test[0].reshape(1,-1))
```

```
array([2])
```

In [ ]:

```
predictions = model.predict(X_test)
```

In [ ]:

```
# Use score method to get accuracy of model
score = model.score(X_test, y_test)
print(score)
```

```
0.9533333333333334
```

In [ ]:

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import metrics
```

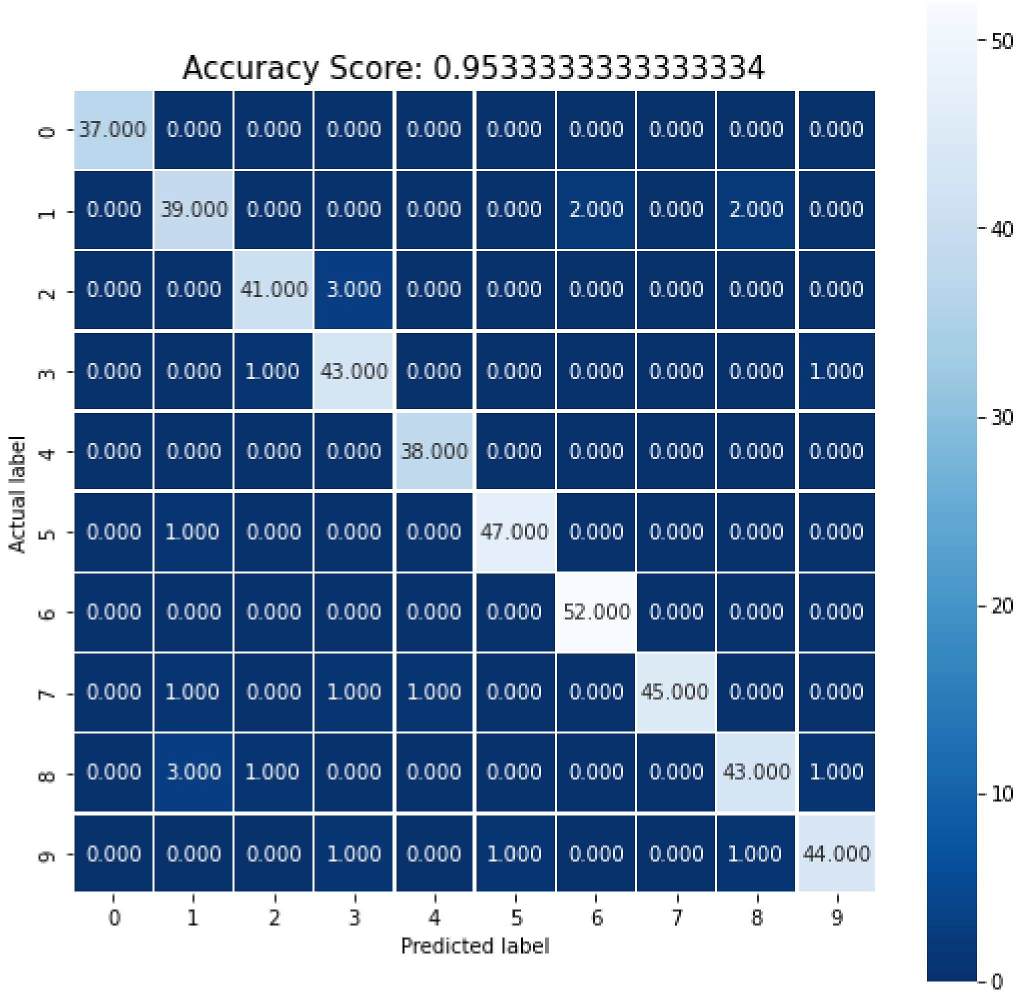
In [ ]:

```
cm = metrics.confusion_matrix(y_test, predictions)
print(cm)
```

```
[[37  0  0  0  0  0  0  0  0  0]
 [ 0 39  0  0  0  0  2  0  2  0]
 [ 0  0 41  3  0  0  0  0  0  0]
 [ 0  0  1 43  0  0  0  0  0  1]
 [ 0  0  0  0 38  0  0  0  0  0]
 [ 0  1  0  0  0 47  0  0  0  0]
 [ 0  0  0  0  0  0 52  0  0  0]
 [ 0  1  0  1  1  0  0 45  0  0]
 [ 0  3  1  0  0  0  0  0 43  1]
 [ 0  0  0  1  0  1  0  0  1 44]]
```

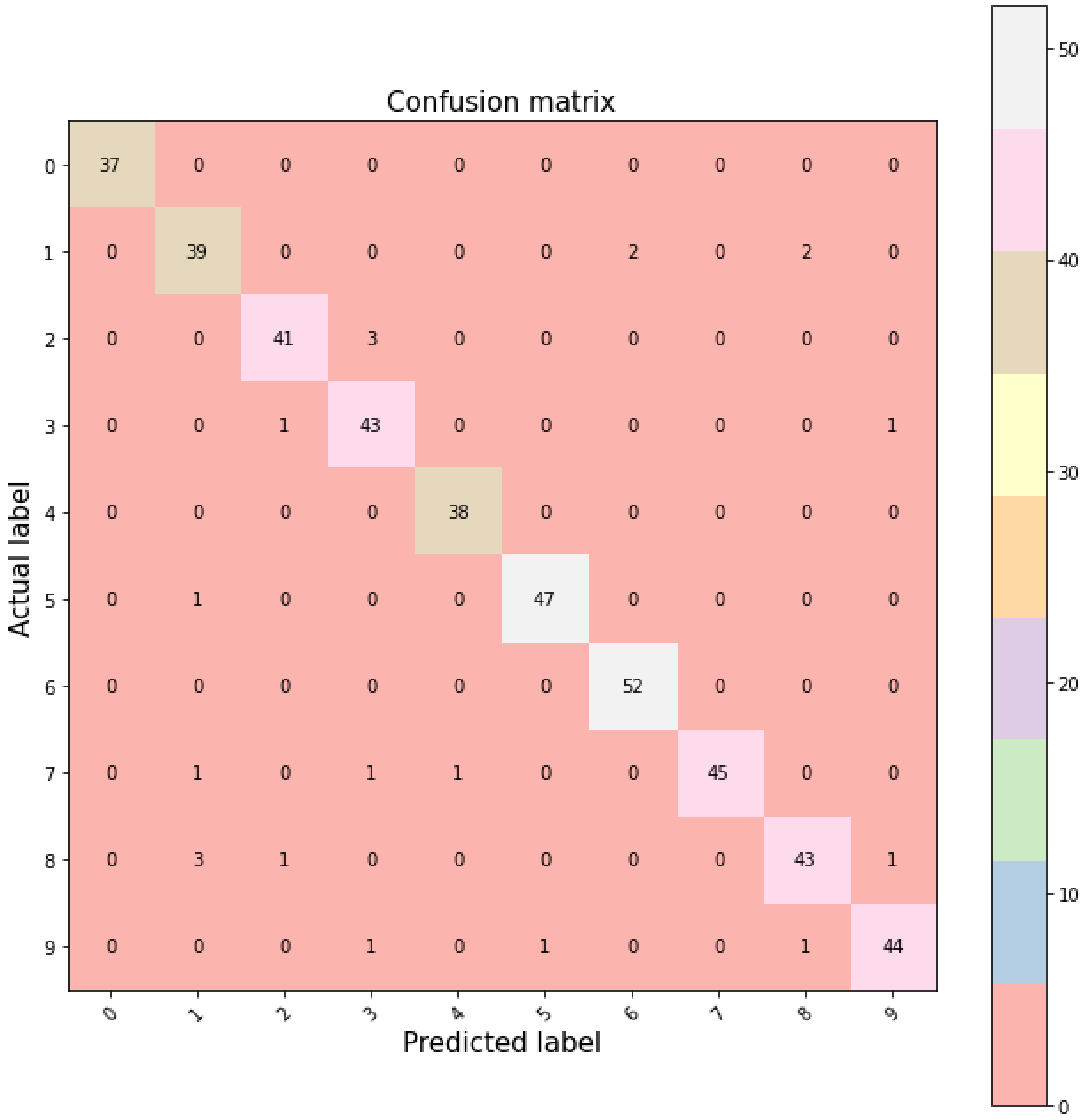
In [ ]:

```
plt.figure(figsize=(9,9))
sns.heatmap(cm, annot=True, fmt=".3f", linewidths=.5, square = True, cmap = 'Blues_r');
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
all_sample_title = 'Accuracy Score: {0}'.format(score)
plt.title(all_sample_title, size = 15);
```



In [ ]:

```
plt.figure(figsize=(9,9))
plt.imshow(cm, interpolation='nearest', cmap='Pastel1')
plt.title('Confusion matrix', size = 15)
plt.colorbar()
tick_marks = np.arange(10)
plt.xticks(tick_marks, ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9"], rotation=45,
plt.yticks(tick_marks, ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9"], size = 10)
plt.tight_layout()
plt.ylabel('Actual label', size = 15)
plt.xlabel('Predicted label', size = 15)
width, height = cm.shape
for x in range(width):
 for y in range(height):
  plt.annotate(str(cm[x][y]), xy=(y, x),
  horizontalalignment='center',
  verticalalignment='center')
```



Display Misclassified images with Predicted Labels Assignment

In [ ]:

```
index = 0
misclassified_images = []
for label, predict in zip(y_test, predictions):
    if label != predict:
        misclassified_images.append(index)
    index +=1
```

In [ ]:

```
print(misclassified_images)
```

```
[37, 94, 109, 124, 130, 181, 196, 211, 218, 235, 251, 301, 312, 331, 335, 378, 398, 413,
416, 425, 440]
```

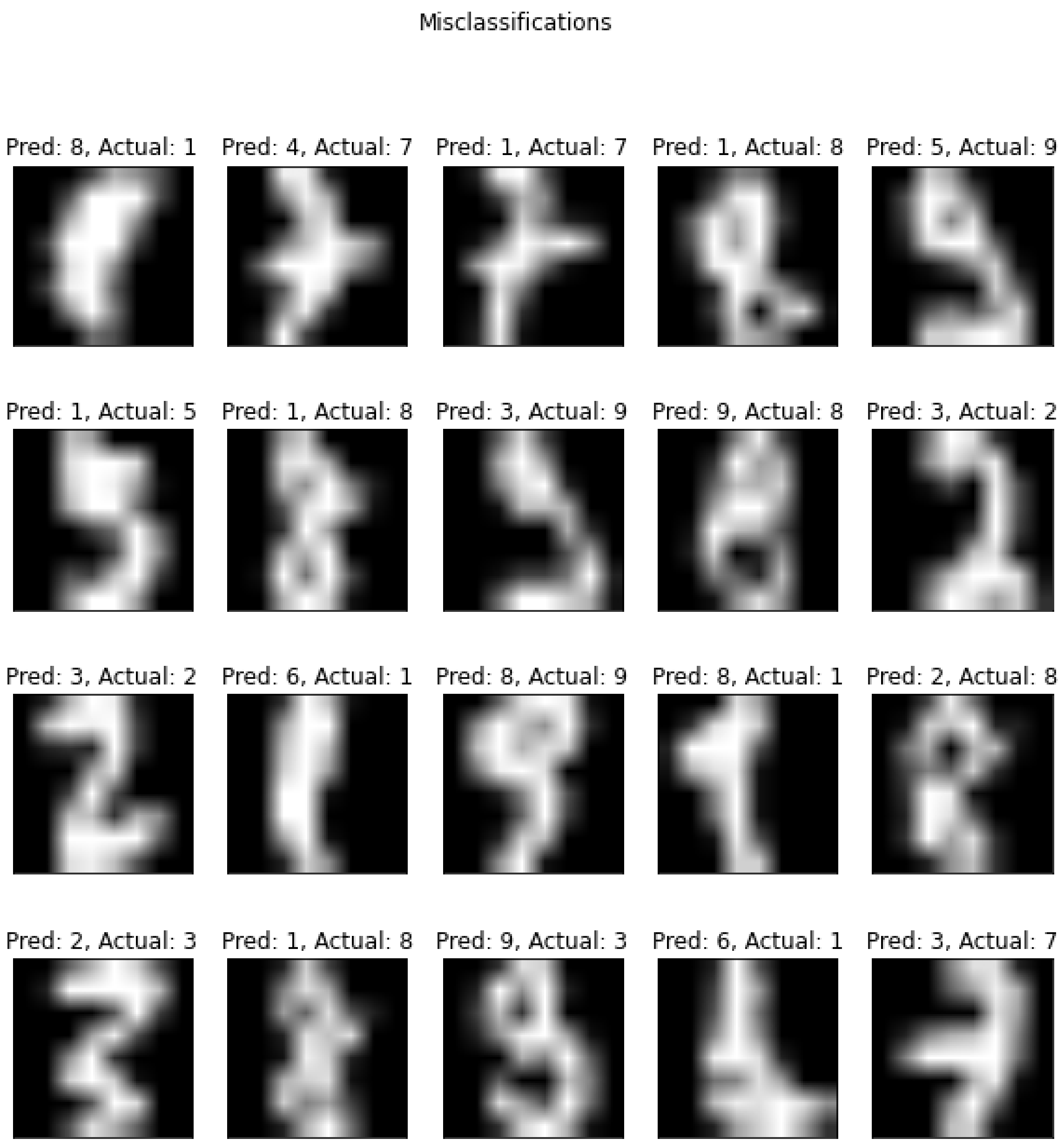
In [ ]:

```
plt.figure(figsize=(10,10))
plt.suptitle('Misclassifications');

for plot_index, bad_index in enumerate(misclassified_images[0:20]):
    p = plt.subplot(4,5, plot_index+1) # 4x5 plot

    p.imshow(X_test[bad_index].reshape(8,8), cmap=plt.cm.gray,
            interpolation='bilinear')
    p.set_xticks(()); p.set_yticks(()) # remove ticks

    p.set_title(f'Pred: {predictions[bad_index]}, Actual: {y_test[bad_index]}');
```



In [ ]: