Use of Logistic Regression on MNIST Classification Problem

use the pre-loaded MNIST Handwritten digit database

Importing libraries

In [5]:

```
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns

# Used for Confusion Matrix
from sklearn import metrics

%matplotlib inline
```

In [7]:

```
digits = load_digits()
```

In [8]:

```
digits.data.shape
```

Out[8]:

```
(1797, 64)
```

In [9]:

```
digits.target.shape
```

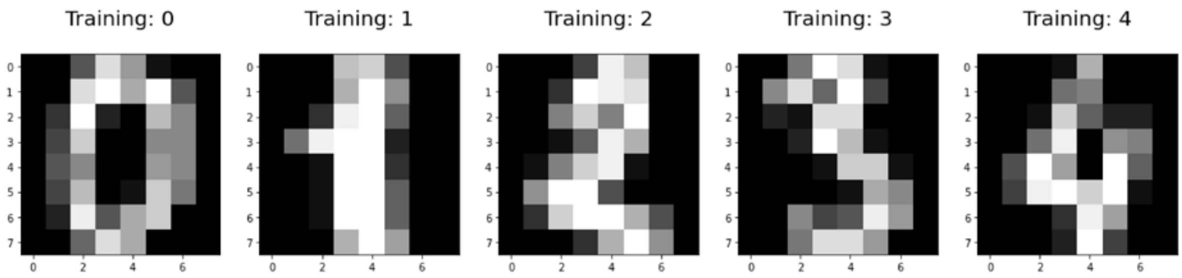
Out[9]:

```
(1797,)
```

Showing the Images and Labels

In [10]:

```
plt.figure(figsize=(20,4))
for index, (image, label) in enumerate(zip(digits.data[0:5], digits.target[0:5])):
    plt.subplot(1, 5, index + 1)
    plt.imshow(np.reshape(image, (8,8)), cmap=plt.cm.gray)
    plt.title('Training: %i\n' % label, fontsize = 20)
```



Splitting Data into Training and Test Sets

In [12]:

```
# test_size: what proportion of original data is used for test set
x_train, x_test, y_train, y_test = train_test_split(
    digits.data, digits.target, test_size=0.25, random_state=0)
```

In [13]:

```
print(x_train.shape)
```

```
(1347, 64)
```

In [14]:

```
print(y_train.shape)
```

```
(1347,)
```

In [15]:

```
print(x_test.shape)
```

```
(450, 64)
```

In [16]:

```
print(y_test.shape)
```

```
(450,)
```

Step 1: Import the model: LogisticRegression

In [18]:

```
from sklearn.linear_model import LogisticRegression
```

Step 2: Make an instance of the Model

In [19]:

```
logisticRegr = LogisticRegression()
```

Step 3: Training the model on the data, storing the information learned from the data

Model is learning the relationship between x (digits) and y (labels)

In [20]:

```
logisticRegr.fit(x_train, y_train)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.p
y:763: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scik
it-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regre
ssion (https://scikit-learn.org/stable/modules/linear_model.html#logistic-re
gression)
  n_iter_i = _check_optimize_result(
```

Out[20]:

```
LogisticRegression()
```

Step 4: Predict the labels of new data (new images)

Uses the information the model learned during the model training process

In [21]:

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

Out[21]:

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

In [22]:

```
# Predict for Multiple Observations (images) at Once
logisticRegr.predict(x_test[0:10])
```

Out[22]:

```
array([2, 8, 2, 6, 6, 7, 1, 9, 8, 5])
```

In [23]:

```
# Make predictions on entire test data
predictions = logisticRegr.predict(x_test)
```

In [24]:

```
predictions.shape
```

Out[24]:

```
(450,)
```

Measuring Model Performance

accuracy (fraction of correct predictions): correct predictions / total number of data points

Basically, how the model performs on new data (test set)

In [25]:

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

```
0.9511111111111111
```

Confusion Matrix (Matplotlib)

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.

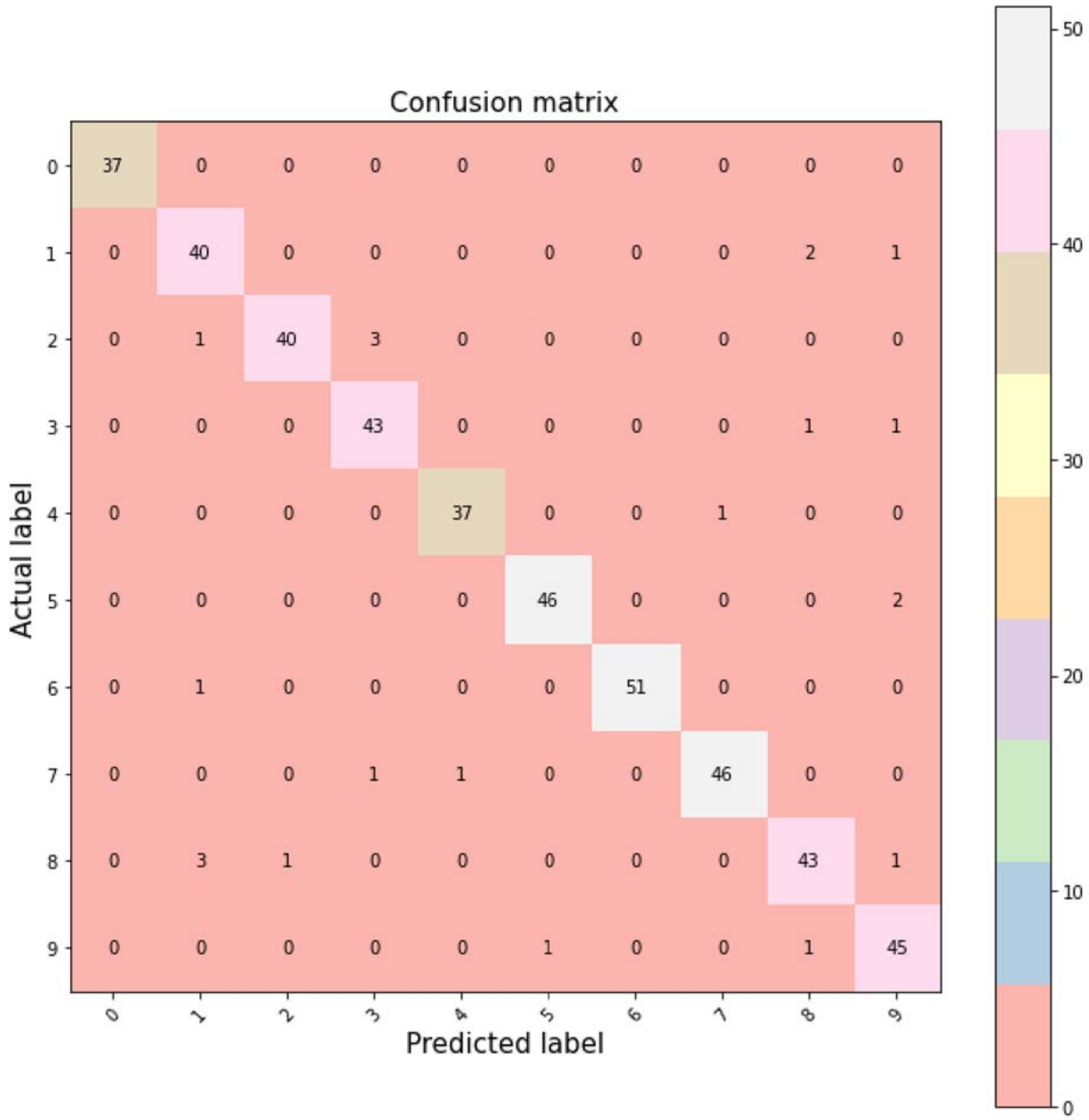
In [29]:

```python
def plot_confusion_matrix(cm, title='Confusion matrix', cmap='Pastel1'):
    plt.figure(figsize=(9,9))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title, 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')
```

In [30]:

```
# confusion matrix
confusion = metrics.confusion_matrix(y_test, predictions)
print('Confusion matrix')
print(confusion)
plt.figure()
plot_confusion_matrix(confusion);
plt.show();
```

```
Confusion matrix
[[37  0  0  0  0  0  0  0  0  0]
 [ 0 40  0  0  0  0  0  0  2  1]
 [ 0  1 40  3  0  0  0  0  0  0]
 [ 0  0  0 43  0  0  0  0  1  1]
 [ 0  0  0  0 37  0  0  1  0  0]
 [ 0  0  0  0  0 46  0  0  0  2]
 [ 0  1  0  0  0  0 51  0  0  0]
 [ 0  0  0  1  1  0  0 46  0  0]
 [ 0  3  1  0  0  0  0  0 43  1]
 [ 0  0  0  0  0  1  0  0  1 45]]

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



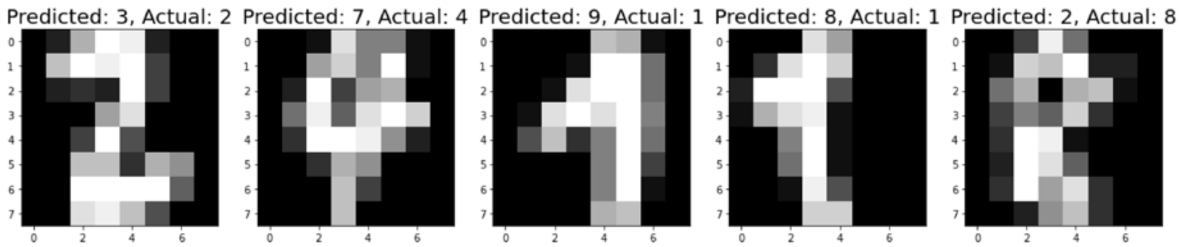
Display Misclassified images with Predicted Labels

In [31]:

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

In [32]:

```
plt.figure(figsize=(20,4))
for plotIndex, wrong in enumerate(misclassifiedIndex[10:15]):
    plt.subplot(1, 5, plotIndex + 1)
    plt.imshow(np.reshape(x_test[wrong], (8,8)), cmap=plt.cm.gray)
    plt.title('Predicted: {}, Actual: {}'.format(predictions[wrong], y_test[wrong]), fontsi
```



In [ ]:

```
```