Recognizing Hand-Written digits

In [7]:
```
#importing necessary libraries
from sklearn.datasets import load_digits
import matplotlib.pyplot as plt
from sklearn import svm ,metrics
from sklearn.model_selection import train_test_split
import numpy as np
import seaborn as sns
```

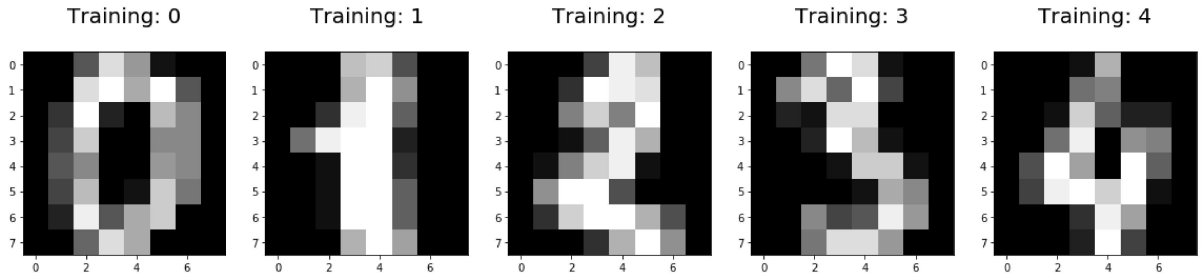
In [8]:
```
%matplotlib inline
digits=load_digits()
```

In [9]:
```
print("Image data shape",digits.data.shape)
print("Label data shape",digits.target.shape)
```

```
Image data shape (1797, 64)
Label data shape (1797,)
```

In [15]:
```
import numpy as np
import matplotlib.pyplot as plt

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)
```



In [ ]:
```
```

Dividing dataset into Training and Test set

In [34]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(digits.data, digits.target, tes
t_size=0.25,random_state=2)
```

In [35]:
```
print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

```
(1347, 64)
(1347,)
(450, 64)
(450,)
```

importing the Logistic Regression model , instantiating it and training it

In [36]:
```
from sklearn.linear_model import LogisticRegression
logisticRegr=LogisticRegression()
logisticRegr.fit(x_train,y_train)
```

```
F:\PROGRAM FILES\lib\site-packages\sklearn\linear_model\_logistic.py:940: Con
vergenceWarning: 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
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regres
sion
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

Out[36]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [37]:
```
print(logisticRegr.predict(x_test[0].reshape(1,-1)))
```

```
[4]
```

In [40]:
```
print(x_test[0].reshape(1,-1))
```

```
[[ 0.  0.  0.  3. 16.  3.  0.  0.  0.  0.  0. 10. 16. 11.  0.  0.  0.  0.
   4. 16. 16.  8.  0.  0.  0.  2. 14. 12. 16.  5.  0.  0.  0. 10. 16. 14.
  16. 16. 11.  0.  0.  5. 12. 13. 16.  8.  3.  0.  0.  0.  0.  2. 15.  3.
   0.  0.  0.  0.  0.  4. 12.  0.  0.  0.]]
```

In [41]:
```
logisticRegr.predict(x_test[0:10])
```

Out[41]:
```
array([4, 0, 9, 1, 8, 7, 1, 5, 1, 6])
```

Predicting for the entire dataset

In [42]:
```
predictions=logisticRegr.predict(x_test)
```

Determing the ACCURACY of the model

In [44]:
```
score=logisticRegr.score(x_test, y_test)
print(score)
```

```
0.9466666666666667
```

In [55]:
```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import metrics
```

In [57]:
```
cm=metrics.confusion_matrix(y_test,predictions)
print(cm)
```

```
[[41  0  0  0  1  0  0  0  0  0]
 [ 0 46  0  1  0  0  0  0  2  1]
 [ 0  0 47  0  0  0  0  0  0  0]
 [ 0  0  0 44  0  1  0  1  2  0]
 [ 0  0  0  0 36  0  0  0  3  1]
 [ 0  1  0  0  1 50  0  0  0  0]
 [ 0  1  0  0  0  0 41  0  1  0]
 [ 0  0  0  0  0  0  0 48  1  0]
 [ 0  0  0  0  0  0  0  0 39  1]
 [ 0  0  0  2  0  1  0  0  2 34]]
```

In [ ]:

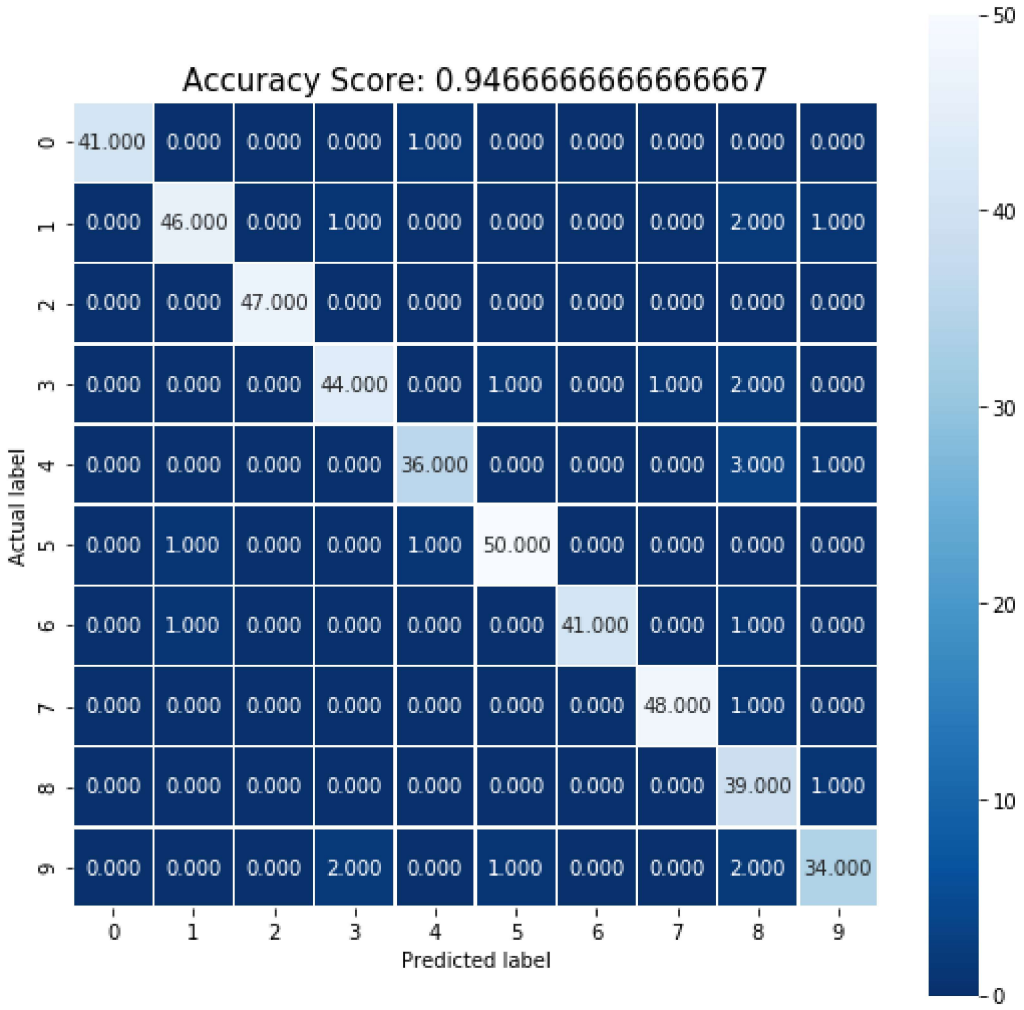
In [ ]:

In [ ]:

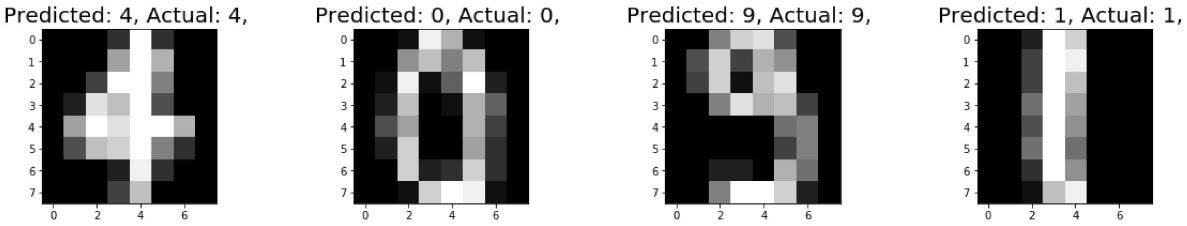
Representing the confusion matrix in a heat map

In [59]:
```
plt.figure(figsize=(9,9))
sns.heatmap(cm, annot=True , fmt=".3f", linewidth=.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)
```

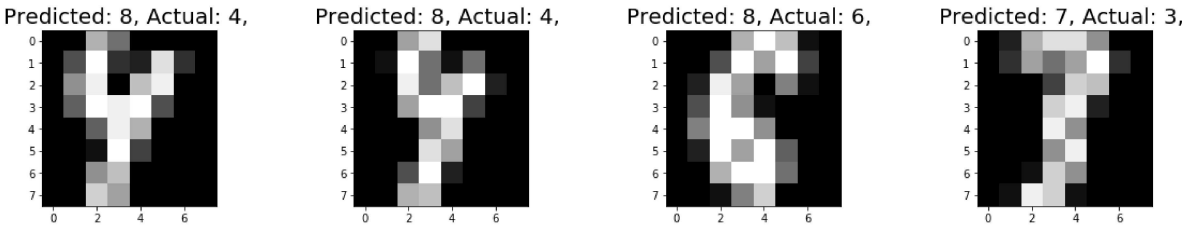
Out[59]: Text(0.5, 1, 'Accuracy Score: 0.9466666666666667')



In [68]:
```
index=0
classifiedIndex=[]
for predict,actual in zip(predictions,y_test):
    if predict==actual:
        classifiedIndex.append(index)
    index+=1
plt.figure(figsize=(20,3))
for plotIndex, wrong in enumerate(classifiedIndex[0:4]):
    plt.subplot(1,4,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]), fontsize=20)
```



In [66]:
```
index=0
misclassifiedIndex=[]
for predict,actual in zip(predictions,y_test):
    if predict!=actual:
        misclassifiedIndex.append(index)
    index+=1
plt.figure(figsize=(20,3))
for plotIndex, wrong in enumerate(misclassifiedIndex[0:4]):
    plt.subplot(1,4,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]), fontsize=20)
```



In [ ]: