Linear Regression (MNIST)

The MNIST database of handwritten digits has training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image.

In [14]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Used for Confusion Matrix
from sklearn import metrics
import seaborn as sns

# Used for Downloading MNIST
#from sklearn.datasets import fetch_mldata

from sklearn.datasets import fetch_openml
dataset = fetch_openml("mnist_784")

# Used for Splitting Training and Test Sets
from sklearn.model_selection import train_test_split

%matplotlib inline
from sklearn.metrics import mean_squared_error, r2_score

from sklearn.linear_model import LinearRegression
```

Downloading MNIST Dataset

In [15]:
```
mnist = fetch_openml("mnist_784")
```

In [16]:
```
mnist
```

Out[16]:
```
{'data': array([[0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        ...,
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.]]),
 'target': array(['5', '0', '4', ..., '4', '5', '6'], dtype=object),
```

```
'frame': None,
'categories': {},
'feature_names': ['pixel1',
 'pixel2',
 'pixel3',
 'pixel4',
 'pixel5',
 'pixel6',
 'pixel7',
 'pixel8',
 'pixel9',
 'pixel10',
 'pixel11',
 'pixel12',
 'pixel13',
 'pixel14',
 'pixel15',
 'pixel16',
 'pixel17',
 'pixel18',
 'pixel19',
 'pixel20',
 'pixel21',
 'pixel22',
 'pixel23',
 'pixel24',
 'pixel25',
 'pixel26',
 'pixel27',
 'pixel28',
 'pixel29',
 'pixel30',
 'pixel31',
 'pixel32',
 'pixel33',
 'pixel34',
 'pixel35',
 'pixel36',
 'pixel37',
 'pixel38',
 'pixel39',
 'pixel40',
 'pixel41',
 'pixel42',
 'pixel43',
 'pixel44',
 'pixel45',
 'pixel46',
 'pixel47',
```

```
'pixel48',
'pixel49',
'pixel50',
'pixel51',
'pixel52',
'pixel53',
'pixel54',
'pixel55',
'pixel56',
'pixel57',
'pixel58',
'pixel59',
'pixel60',
'pixel61',
'pixel62',
'pixel63',
'pixel64',
'pixel65',
'pixel66',
'pixel67',
'pixel68',
'pixel69',
'pixel70',
'pixel71',
'pixel72',
'pixel73',
'pixel74',
'pixel75',
'pixel76',
'pixel77',
'pixel78',
'pixel79',
'pixel80',
'pixel81',
'pixel82',
'pixel83',
'pixel84',
'pixel85',
'pixel86',
'pixel87',
'pixel88',
'pixel89',
'pixel90',
'pixel91',
'pixel92',
'pixel93',
'pixel94',
'pixel95',
'pixel96',
```

```
 'pixel97',
 'pixel98',
 'pixel99',
 'pixel100',
 'pixel101',
 'pixel102',
 'pixel103',
 'pixel104',
 'pixel105',
 'pixel106',
 'pixel107',
 'pixel108',
 'pixel109',
 'pixel110',
 'pixel111',
 'pixel112',
 'pixel113',
 'pixel114',
 'pixel115',
 'pixel116',
 'pixel117',
 'pixel118',
 'pixel119',
 'pixel120',
 'pixel121',
 'pixel122',
 'pixel123',
 'pixel124',
 'pixel125',
 'pixel126',
 'pixel127',
 'pixel128',
 'pixel129',
 'pixel130',
 'pixel131',
 'pixel132',
 'pixel133',
 'pixel134',
 'pixel135',
 'pixel136',
 'pixel137',
 'pixel138',
 'pixel139',
 'pixel140',
 'pixel141',
 'pixel142',
 'pixel143',
 'pixel144',
 'pixel145',
```

```
'pixel146',
'pixel147',
'pixel148',
'pixel149',
'pixel150',
'pixel151',
'pixel152',
'pixel153',
'pixel154',
'pixel155',
'pixel156',
'pixel157',
'pixel158',
'pixel159',
'pixel160',
'pixel161',
'pixel162',
'pixel163',
'pixel164',
'pixel165',
'pixel166',
'pixel167',
'pixel168',
'pixel169',
'pixel170',
'pixel171',
'pixel172',
'pixel173',
'pixel174',
'pixel175',
'pixel176',
'pixel177',
'pixel178',
'pixel179',
'pixel180',
'pixel181',
'pixel182',
'pixel183',
'pixel184',
'pixel185',
'pixel186',
'pixel187',
'pixel188',
'pixel189',
'pixel190',
'pixel191',
'pixel192',
'pixel193',
'pixel194',
```

```
 'pixel195',
 'pixel196',
 'pixel197',
 'pixel198',
 'pixel199',
 'pixel200',
 'pixel201',
 'pixel202',
 'pixel203',
 'pixel204',
 'pixel205',
 'pixel206',
 'pixel207',
 'pixel208',
 'pixel209',
 'pixel210',
 'pixel211',
 'pixel212',
 'pixel213',
 'pixel214',
 'pixel215',
 'pixel216',
 'pixel217',
 'pixel218',
 'pixel219',
 'pixel220',
 'pixel221',
 'pixel222',
 'pixel223',
 'pixel224',
 'pixel225',
 'pixel226',
 'pixel227',
 'pixel228',
 'pixel229',
 'pixel230',
 'pixel231',
 'pixel232',
 'pixel233',
 'pixel234',
 'pixel235',
 'pixel236',
 'pixel237',
 'pixel238',
 'pixel239',
 'pixel240',
 'pixel241',
 'pixel242',
 'pixel243',
```

```
'pixel244',
'pixel245',
'pixel246',
'pixel247',
'pixel248',
'pixel249',
'pixel250',
'pixel251',
'pixel252',
'pixel253',
'pixel254',
'pixel255',
'pixel256',
'pixel257',
'pixel258',
'pixel259',
'pixel260',
'pixel261',
'pixel262',
'pixel263',
'pixel264',
'pixel265',
'pixel266',
'pixel267',
'pixel268',
'pixel269',
'pixel270',
'pixel271',
'pixel272',
'pixel273',
'pixel274',
'pixel275',
'pixel276',
'pixel277',
'pixel278',
'pixel279',
'pixel280',
'pixel281',
'pixel282',
'pixel283',
'pixel284',
'pixel285',
'pixel286',
'pixel287',
'pixel288',
'pixel289',
'pixel290',
'pixel291',
'pixel292',
```

```
'pixel293',
'pixel294',
'pixel295',
'pixel296',
'pixel297',
'pixel298',
'pixel299',
'pixel300',
'pixel301',
'pixel302',
'pixel303',
'pixel304',
'pixel305',
'pixel306',
'pixel307',
'pixel308',
'pixel309',
'pixel310',
'pixel311',
'pixel312',
'pixel313',
'pixel314',
'pixel315',
'pixel316',
'pixel317',
'pixel318',
'pixel319',
'pixel320',
'pixel321',
'pixel322',
'pixel323',
'pixel324',
'pixel325',
'pixel326',
'pixel327',
'pixel328',
'pixel329',
'pixel330',
'pixel331',
'pixel332',
'pixel333',
'pixel334',
'pixel335',
'pixel336',
'pixel337',
'pixel338',
'pixel339',
'pixel340',
'pixel341',
```

```
'pixel342',
'pixel343',
'pixel344',
'pixel345',
'pixel346',
'pixel347',
'pixel348',
'pixel349',
'pixel350',
'pixel351',
'pixel352',
'pixel353',
'pixel354',
'pixel355',
'pixel356',
'pixel357',
'pixel358',
'pixel359',
'pixel360',
'pixel361',
'pixel362',
'pixel363',
'pixel364',
'pixel365',
'pixel366',
'pixel367',
'pixel368',
'pixel369',
'pixel370',
'pixel371',
'pixel372',
'pixel373',
'pixel374',
'pixel375',
'pixel376',
'pixel377',
'pixel378',
'pixel379',
'pixel380',
'pixel381',
'pixel382',
'pixel383',
'pixel384',
'pixel385',
'pixel386',
'pixel387',
'pixel388',
'pixel389',
'pixel390',
```

```
'pixel391',
'pixel392',
'pixel393',
'pixel394',
'pixel395',
'pixel396',
'pixel397',
'pixel398',
'pixel399',
'pixel400',
'pixel401',
'pixel402',
'pixel403',
'pixel404',
'pixel405',
'pixel406',
'pixel407',
'pixel408',
'pixel409',
'pixel410',
'pixel411',
'pixel412',
'pixel413',
'pixel414',
'pixel415',
'pixel416',
'pixel417',
'pixel418',
'pixel419',
'pixel420',
'pixel421',
'pixel422',
'pixel423',
'pixel424',
'pixel425',
'pixel426',
'pixel427',
'pixel428',
'pixel429',
'pixel430',
'pixel431',
'pixel432',
'pixel433',
'pixel434',
'pixel435',
'pixel436',
'pixel437',
'pixel438',
'pixel439',
```

```
'pixel440',
'pixel441',
'pixel442',
'pixel443',
'pixel444',
'pixel445',
'pixel446',
'pixel447',
'pixel448',
'pixel449',
'pixel450',
'pixel451',
'pixel452',
'pixel453',
'pixel454',
'pixel455',
'pixel456',
'pixel457',
'pixel458',
'pixel459',
'pixel460',
'pixel461',
'pixel462',
'pixel463',
'pixel464',
'pixel465',
'pixel466',
'pixel467',
'pixel468',
'pixel469',
'pixel470',
'pixel471',
'pixel472',
'pixel473',
'pixel474',
'pixel475',
'pixel476',
'pixel477',
'pixel478',
'pixel479',
'pixel480',
'pixel481',
'pixel482',
'pixel483',
'pixel484',
'pixel485',
'pixel486',
'pixel487',
'pixel488',
```

```
'pixel489',
'pixel490',
'pixel491',
'pixel492',
'pixel493',
'pixel494',
'pixel495',
'pixel496',
'pixel497',
'pixel498',
'pixel499',
'pixel500',
'pixel501',
'pixel502',
'pixel503',
'pixel504',
'pixel505',
'pixel506',
'pixel507',
'pixel508',
'pixel509',
'pixel510',
'pixel511',
'pixel512',
'pixel513',
'pixel514',
'pixel515',
'pixel516',
'pixel517',
'pixel518',
'pixel519',
'pixel520',
'pixel521',
'pixel522',
'pixel523',
'pixel524',
'pixel525',
'pixel526',
'pixel527',
'pixel528',
'pixel529',
'pixel530',
'pixel531',
'pixel532',
'pixel533',
'pixel534',
'pixel535',
'pixel536',
'pixel537',
```

```
'pixel538',
'pixel539',
'pixel540',
'pixel541',
'pixel542',
'pixel543',
'pixel544',
'pixel545',
'pixel546',
'pixel547',
'pixel548',
'pixel549',
'pixel550',
'pixel551',
'pixel552',
'pixel553',
'pixel554',
'pixel555',
'pixel556',
'pixel557',
'pixel558',
'pixel559',
'pixel560',
'pixel561',
'pixel562',
'pixel563',
'pixel564',
'pixel565',
'pixel566',
'pixel567',
'pixel568',
'pixel569',
'pixel570',
'pixel571',
'pixel572',
'pixel573',
'pixel574',
'pixel575',
'pixel576',
'pixel577',
'pixel578',
'pixel579',
'pixel580',
'pixel581',
'pixel582',
'pixel583',
'pixel584',
'pixel585',
'pixel586',
```

```
'pixel587',
'pixel588',
'pixel589',
'pixel590',
'pixel591',
'pixel592',
'pixel593',
'pixel594',
'pixel595',
'pixel596',
'pixel597',
'pixel598',
'pixel599',
'pixel600',
'pixel601',
'pixel602',
'pixel603',
'pixel604',
'pixel605',
'pixel606',
'pixel607',
'pixel608',
'pixel609',
'pixel610',
'pixel611',
'pixel612',
'pixel613',
'pixel614',
'pixel615',
'pixel616',
'pixel617',
'pixel618',
'pixel619',
'pixel620',
'pixel621',
'pixel622',
'pixel623',
'pixel624',
'pixel625',
'pixel626',
'pixel627',
'pixel628',
'pixel629',
'pixel630',
'pixel631',
'pixel632',
'pixel633',
'pixel634',
'pixel635',
```

```
'pixel636',
'pixel637',
'pixel638',
'pixel639',
'pixel640',
'pixel641',
'pixel642',
'pixel643',
'pixel644',
'pixel645',
'pixel646',
'pixel647',
'pixel648',
'pixel649',
'pixel650',
'pixel651',
'pixel652',
'pixel653',
'pixel654',
'pixel655',
'pixel656',
'pixel657',
'pixel658',
'pixel659',
'pixel660',
'pixel661',
'pixel662',
'pixel663',
'pixel664',
'pixel665',
'pixel666',
'pixel667',
'pixel668',
'pixel669',
'pixel670',
'pixel671',
'pixel672',
'pixel673',
'pixel674',
'pixel675',
'pixel676',
'pixel677',
'pixel678',
'pixel679',
'pixel680',
'pixel681',
'pixel682',
'pixel683',
'pixel684',
```

```
'pixel685',
'pixel686',
'pixel687',
'pixel688',
'pixel689',
'pixel690',
'pixel691',
'pixel692',
'pixel693',
'pixel694',
'pixel695',
'pixel696',
'pixel697',
'pixel698',
'pixel699',
'pixel700',
'pixel701',
'pixel702',
'pixel703',
'pixel704',
'pixel705',
'pixel706',
'pixel707',
'pixel708',
'pixel709',
'pixel710',
'pixel711',
'pixel712',
'pixel713',
'pixel714',
'pixel715',
'pixel716',
'pixel717',
'pixel718',
'pixel719',
'pixel720',
'pixel721',
'pixel722',
'pixel723',
'pixel724',
'pixel725',
'pixel726',
'pixel727',
'pixel728',
'pixel729',
'pixel730',
'pixel731',
'pixel732',
'pixel733',
```

```
'pixel734',
'pixel735',
'pixel736',
'pixel737',
'pixel738',
'pixel739',
'pixel740',
'pixel741',
'pixel742',
'pixel743',
'pixel744',
'pixel745',
'pixel746',
'pixel747',
'pixel748',
'pixel749',
'pixel750',
'pixel751',
'pixel752',
'pixel753',
'pixel754',
'pixel755',
'pixel756',
'pixel757',
'pixel758',
'pixel759',
'pixel760',
'pixel761',
'pixel762',
'pixel763',
'pixel764',
'pixel765',
'pixel766',
'pixel767',
'pixel768',
'pixel769',
'pixel770',
'pixel771',
'pixel772',
'pixel773',
'pixel774',
'pixel775',
'pixel776',
'pixel777',
'pixel778',
'pixel779',
'pixel780',
'pixel781',
'pixel782',
```

```
  'pixel783',
  'pixel784'],
 'target_names': ['class'],
 'DESCR': "**Author**: Yann LeCun, Corinna Cortes, Christopher J.C. Burges  \n**Source**: [MNIST Website](htt
p://yann.lecun.com/exdb/mnist/) - Date unknown  \n**Please cite**:  \n\nThe MNIST database of handwritten digit
s with 784 features, raw data available at: http://yann.lecun.com/exdb/mnist/. It can be split in a training se
t of the first 60,000 examples, and a test set of 10,000 examples  \n\nIt is a subset of a larger set available
from NIST. The digits have been size-normalized and centered in a fixed-size image. It is a good database for p
eople who want to try learning techniques and pattern recognition methods on real-world data while spending min
imal efforts on preprocessing and formatting. The original black and white (bilevel) images from NIST were size
normalized to fit in a 20x20 pixel box while preserving their aspect ratio. The resulting images contain grey l
evels as a result of the anti-aliasing technique used by the normalization algorithm. the images were centered
in a 28x28 image by computing the center of mass of the pixels, and translating the image so as to position thi
s point at the center of the 28x28 field.  \n\nWith some classification methods (particularly template-based me
thods, such as SVM and K-nearest neighbors), the error rate improves when the digits are centered by bounding b
ox rather than center of mass. If you do this kind of pre-processing, you should report it in your publication
s. The MNIST database was constructed from NIST's NIST originally designated SD-3 as their training set and SD-
1 as their test set. However, SD-3 is much cleaner and easier to recognize than SD-1. The reason for this can b
e found on the fact that SD-3 was collected among Census Bureau employees, while SD-1 was collected among high-
school students. Drawing sensible conclusions from learning experiments requires that the result be independent
of the choice of training set and test among the complete set of samples. Therefore it was necessary to build a
new database by mixing NIST's datasets.  \n\nThe MNIST training set is composed of 30,000 patterns from SD-3 an
d 30,000 patterns from SD-1. Our test set was composed of 5,000 patterns from SD-3 and 5,000 patterns from SD-
1. The 60,000 pattern training set contained examples from approximately 250 writers. We made sure that the set
s of writers of the training set and test set were disjoint. SD-1 contains 58,527 digit images written by 500 d
ifferent writers. In contrast to SD-3, where blocks of data from each writer appeared in sequence, the data in
SD-1 is scrambled. Writer identities for SD-1 is available and we used this information to unscramble the write
rs. We then split SD-1 in two: characters written by the first 250 writers went into our new training set. The
remaining 250 writers were placed in our test set. Thus we had two sets with nearly 30,000 examples each. The n
ew training set was completed with enough examples from SD-3, starting at pattern # 0, to make a full set of 6
0,000 training patterns. Similarly, the new test set was completed with SD-3 examples starting at pattern # 35,
000 to make a full set with 60,000 test patterns. Only a subset of 10,000 test images (5,000 from SD-1 and 5,00
0 from SD-3) is available on this site. The full 60,000 sample training set is available.\n\nDownloaded from op
enml.org.",
 'details': {'id': '554',
  'name': 'mnist_784',
  'version': '1',
  'format': 'ARFF',
  'upload_date': '2014-09-29T03:28:38',
  'licence': 'Public',
  'url': 'https://www.openml.org/data/v1/download/52667/mnist_784.arff',
  'file_id': '52667',
  'default_target_attribute': 'class',
  'tag': ['AzurePilot',
   'OpenML-CC18',
   'OpenML100',
   'study_1',
   'study_123',
   'study_41',
```

```
   'study_99',
   'vision'],
  'visibility': 'public',
  'status': 'active',
  'processing_date': '2018-10-03 21:23:30',
  'md5_checksum': '0298d579eb1b86163de7723944c7e495'},
 'url': 'https://www.openml.org/d/554'}
```


In [17]:
```python
# These are the images
mnist.data.shape
```

Out[17]:
```
(70000, 784)
```

In [18]:
```python
# These are the labels
mnist.target.shape
```

Out[18]:
```
(70000,)
```

Splitting Data into Training and Test Sets

In [19]:
```python
# test_size:
train_img, test_img, train_lbl, test_lbl = train_test_split(
    mnist.data, mnist.target, test_size=1/7.0, random_state=0)
```

In [20]:
```python
type(train_img)
```

Out[20]:
```
numpy.ndarray
```

In [21]:
```python
acc_list = []
for i in [0,1,2,3,4]:

    zero_four_train= [train_img [key] for (key, label) in enumerate(train_lbl) if int(label) == i or int(label)
    zero_four_label = [train_lbl[key] for (key, label) in enumerate(train_lbl) if int(label) == i or int(label)
    zero_four_test= [test_img [key] for (key, label) in enumerate(test_lbl) if int(label) == i or int(label) ==
    zero_four_test_label = [test_lbl[key] for (key, label) in enumerate(test_lbl) if int(label) == i or int(lab

    print(f'{type(zero_four_test_label)}')


    zero_four_train = np.array(zero_four_train)

    zero_four_label = np.array(zero_four_label)
```

```
zero_four_test = np.array(zero_four_test)
zero_four_test_label = np.array(zero_four_test_label)
```

```
print(f'Shape {zero_four_label}')
train=zero_four_train[0:4000]
label=zero_four_label[0:4000]

logisticRegr = LinearRegression()

np.set_printoptions(precision=2, suppress=True)
logisticRegr.fit(zero_four_train[0:4000], zero_four_label[0:4000])
print('intercept ----->:', logisticRegr.intercept_)
p = logisticRegr.coef_[0]
print(f"Weight ------>:  {p}")

z = zero_four_test[0:2000]

predictions = logisticRegr.predict(z)

print(f"prediction----> {predictions} ")

p = np.round(predictions).astype(int)


print(f"Prediction ----> {p} ")

print(f"zero_four_test_label----> {pd.to_numeric(zero_four_test_label[0:2000])} ")


zero_ = zero_four_test_label[0:2000].astype(np.int)

print(f"zero_four_test_label----> { zero_} ")

from sklearn.metrics import accuracy_score

acc = accuracy_score(zero_,p)

acc_list.append(acc)
```

```
    print(f' Accuracy -----> {acc}')


std_dev = acc_list
p = round(np.std(std_dev, dtype=np.float64),4)
print(f' Standard Error ----> {p}')
```

```
<class 'list'>
Shape ['0' '0' '0' ... '0' '0' '4']
intercept ----->: 1.871914749230491
Weight ------>:  -225609.32392792773
prediction----> [ 0.48  3.83  4.   ... -0.79  4.24  0.01]
Prediction ----> [ 0  4  4 ... -1  4  0]
zero_four_test_label----> [0 4 4 ... 0 4 0]
zero_four_test_label----> [0 4 4 ... 0 4 0]
 Accuracy -----> 0.6634320735444331
<class 'list'>
Shape ['1' '5' '5' ... '1' '1' '1']
intercept ----->: 4.102126213801093
Weight ------>:  425407.34558768675
prediction----> [1.   1.24 1.2  ... 3.72 4.03 1.95]
Prediction ----> [1 1 1 ... 4 4 2]
zero_four_test_label----> [1 1 1 ... 5 5 1]
zero_four_test_label----> [1 1 1 ... 5 5 1]
 Accuracy -----> 0.7665
<class 'list'>
Shape ['2' '6' '2' ... '6' '2' '6']
intercept ----->: 3.797575014773756
Weight ------>:  -1402430.064593433
prediction----> [2.48 2.33 4.38 ... 1.76 4.95 1.89]
Prediction ----> [2 2 4 ... 2 5 2]
zero_four_test_label----> [2 2 6 ... 2 6 2]
zero_four_test_label----> [2 2 6 ... 2 6 2]
 Accuracy -----> 0.5335
<class 'list'>
Shape ['7' '3' '7' ... '3' '7' '7']
intercept ----->: 5.612925636491273
Weight ------>:  378931.6271637447
prediction----> [6.96 7.48 5.62 ... 7.5  4.63 6.96]
Prediction ----> [7 7 6 ... 7 5 7]
zero_four_test_label----> [7 7 7 ... 7 3 7]
zero_four_test_label----> [7 7 7 ... 7 3 7]
 Accuracy -----> 0.563
<class 'list'>
Shape ['8' '8' '8' ... '8' '4' '8']
intercept ----->: 5.678687956075184
Weight ------>:  -313486.29262392083
prediction----> [4.07 4.36 4.17 ... 3.69 7.78 6.79]
```

```
Prediction ----> [4 4 4 ... 4 8 7]
zero_four_test_label----> [4 4 4 ... 4 8 8]
zero_four_test_label----> [4 4 4 ... 4 8 8]
 Accuracy -----> 0.5641558441558442
 Standard Error ----> 0.0862
```

In [ ]:

In [22]:
```
print(train_img.shape)
```

```
(60000, 784)
```

In [23]:
```
print(train_lbl.shape)
```

```
(60000,)
```

In [24]:
```
print(test_img.shape)
```

```
(10000, 784)
```

In [25]:
```
print(test_lbl.shape)
```

```
(10000,)
```

In [26]:
```
zero_four_train= [train_img [key] for (key, label) in enumerate(train_lbl) if int(label) == 3 or int(label) ==
```

In [27]:
```
zero_four_label = [label for (key, label) in enumerate(train_lbl) if int(label) == 3 or int(label) == 4]
```

In [28]:
```
print(train_img.shape)
```

```
(60000, 784)
```

In [29]:
```
train_img = train_img[0:2000]
train_lbl = train_lbl[0:2000]
test_img = test_img[0:2000]
test_lbl = test_lbl[0:2000]
```

In [30]:
```
print(len(train_img))
```

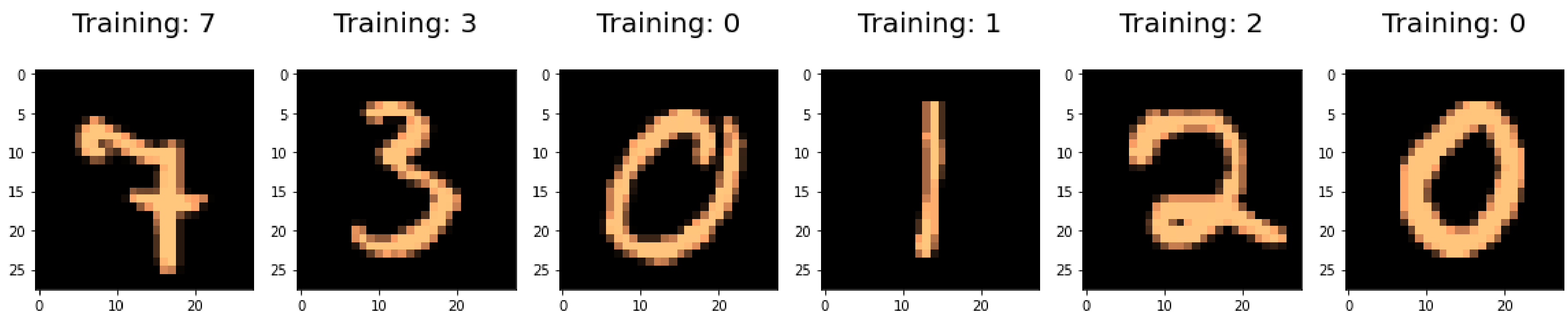
```
2000
```

Showing Training Digits and Labels

In [ ]:

In [31]:

```python
plt.figure(figsize=(20,4))
for index, (image, label) in enumerate(zip(train_img[0:6], train_lbl[0:6])):
    plt.subplot(1, 6, index + 1)
    plt.imshow(np.reshape(image, (28,28)), cmap=plt.cm.copper)
    plt.title('Training: %s\n' % label, fontsize = 20)
```



In [32]:

```python
# This is how the computer sees the number 4
print(train_img[0])
```

```
[   0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
    0.    0.    0.    0.    0.    0.   40.  195.  126.    6.    0.    0.    0.    0.
    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
    0.    0.    0.    0.    0.   52.  222.  254.  254.  203.   77.    5.    0.    0.
    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
    0.    0.    0.    0.    0.  111.  254.  254.  254.  254.  254.  205.   65.    0.
    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
    0.    0.    0.    0.    0.  111.  254.  247.   86.  127.  222.  254.  242.  201.
   79.   21.   66.  159.   62.    0.    0.    0.    0.    0.    0.    0.    0.    0.
    0.    0.    0.    0.    0.   39.  228.  254.  254.   94.   58.  186.  254.  254.
  254.  254.  254.  254.  129.    0.    0.    0.    0.    0.    0.    0.    0.    0.
    0.    0.    0.    0.    0.    0.   16.  104.   94.   22.    0.    4.   25.  142.
  224.  231.  254.  254.  129.    0.    0.    0.    0.    0.    0.    0.    0.    0.
    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
    0.   17.  226.  254.  129.    0.    0.    0.    0.    0.    0.    0.    0.    0.
    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
```

```
   0.  48. 246. 254. 129.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.  47. 245. 254. 129.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0. 116.  92.
  90.  99. 254. 254. 154.  15.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.  16. 215. 254.
 254. 254. 254. 254. 254. 220. 205. 166.   4.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0. 113.
 195. 254. 254. 254. 254. 254. 209.  24.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   1. 175. 254. 254. 104.  40.  19.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0. 132. 254. 254.  30.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0. 100. 254. 254.  30.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.  61. 254. 254.  30.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.  61. 254. 254.  30.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.  61. 254. 254.  30.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.  26. 232. 254.  30.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0. 176. 160.   7.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.]
```

In [33]:

```python
print(train_lbl[0])
```

```
7
```

Using Linear Regression on Entire Dataset

In [34]:

```python
from sklearn.linear_model import LinearRegression

logisticRegr = LinearRegression()
logisticRegr.fit(train_img, train_lbl)
```

Out[34]:

```
LinearRegression()
```

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

In [35]:

```python
np.set_printoptions(precision=2, suppress=True)
p = str(round(logisticRegr.intercept_, 2))
```

```
print('intercept:', p)

p = logisticRegr.coef_[0]

#p = np.array2string(p)
p = str(round(p, 2))
print(f"{p}")
```

```
intercept: 3.28
-2657505.35
```

Uses the information the model learned during the model training process

In [36]:
```
# Returns a NumPy Array
# Predict for One Observation (image)
lin = logisticRegr.predict(test_img[0].reshape(1,-1))

print(f' predicted --->{lin} , actual----> {test_lbl[0]}')
```

```
 predicted --->[2.92] , actual----> 0
```

Measuring Model Performance

In [37]:
```
np.set_printoptions(precision=2, suppress=True)
p = str(round(logisticRegr.intercept_, 2))
print('intercept:', p)

p = logisticRegr.coef_[0]

#p = np.array2string(p)
print(f"{p}")
```

```
intercept: 3.28
-2657505.34562636
```

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

In [40]:
```
logisticRegr.predict(test_img[0:10])
```

Out[40]:
```
array([2.92, 4.58, 1.95, 1.95, 1.98, 4.27, 7.06, 3.73, 2.05, 8.87])
```

In [41]:
```
print(f'{test_lbl[0:10]}')
```

```
['0' '4' '1' '2' '7' '9' '7' '1' '1' '7']
```

In [42]:
```
# Make predictions on test data
```

```python
predictions = logisticRegr.predict(test_img[0:10])

print(f' pred {predictions} ')


acc = mean_squared_error(predictions,test_lbl[0:10])

print(f'{ acc:.2f}')
```

```
 pred [2.92 4.58 1.95 1.95 1.98 4.27 7.06 3.73 2.05 8.87]
6.94
```

In [43]:

```python
score = logisticRegr.score(test_img[0:10], test_lbl[0:10])
print(score)
```

```
0.2982933200944149
```

In [44]:

```python
# Make predictions on test data
predictions = logisticRegr.predict(test_img)



print(f'{predictions}')
```

```
[2.92 4.58 1.95 ... 2.67 7.27 6.93]
```

Display Misclassified images with Predicted Labels

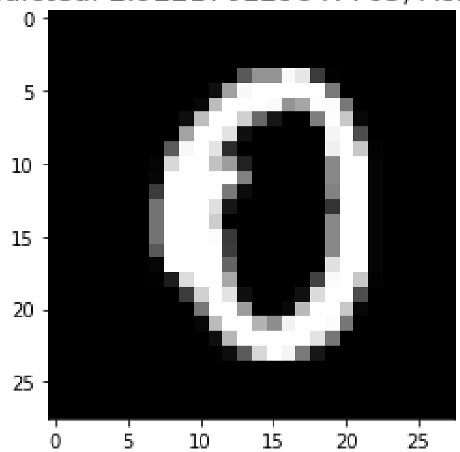
In [45]:

```python
index = 0
misclassifiedIndexes = []
for label, predict in zip(test_lbl, predictions):
    predict = np.round(predict)
    if label != predict:
        misclassifiedIndexes.append(index)
    index +=1
```

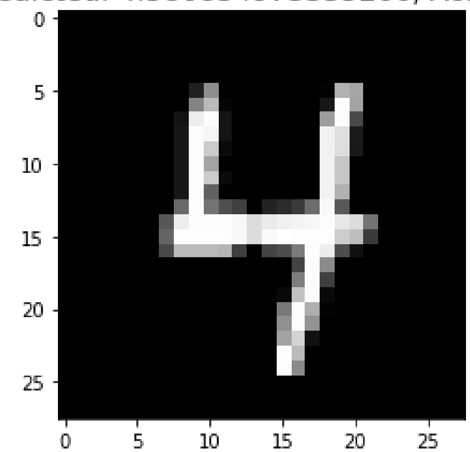
In [46]:

```python
plt.figure(figsize=(20,4))
for plotIndex, badIndex in enumerate(misclassifiedIndexes[0:2]):
    plt.subplot(1, 2, plotIndex + 1)
    plt.imshow(np.reshape(test_img[badIndex], (28,28)), cmap=plt.cm.gray)
    plt.title('Predicted: {}, Actual: {}'.format(predictions[badIndex], test_lbl[badIndex]), fontsize = 15)
```

Predicted: 2.922176129847765, Actual: 0



Predicted: 4.580834973335266, Actual: 4



In [ ]:

In [ ]:

In [ ]: