Use of Support Vector Machine Models on MNIST Classification Problem

use the MNIST Handwritten digit database, Dataset link: http://yann.lecun.com/exdb/mnist/ (http://yann.lecun.com/exdb/mnist/)

https://www.kaggle.com/c/digit-recognizer/data

Importing libraries

In [22]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import linear_model
from sklearn.model_selection import train_test_split
import gc
```

In [23]:

```
# read the dataset
digits = pd.read_csv("train.csv")
digits.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 42000 entries, 0 to 41999
Columns: 785 entries, label to pixel783
dtypes: int64(785)
memory usage: 251.5 MB
```

In [24]:

```
digits.head()
```

Out[24]:

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	...	pixel774	pixel
0	1	0	0	0	0	0	0	0	0	0	...	0	
1	0	0	0	0	0	0	0	0	0	0	...	0	
2	1	0	0	0	0	0	0	0	0	0	...	0	
3	4	0	0	0	0	0	0	0	0	0	...	0	
4	0	0	0	0	0	0	0	0	0	0	...	0	

5 rows × 785 columns

In [25]:

```
four = digits.iloc[3, 1:]
four.shape
```

Out[25]:

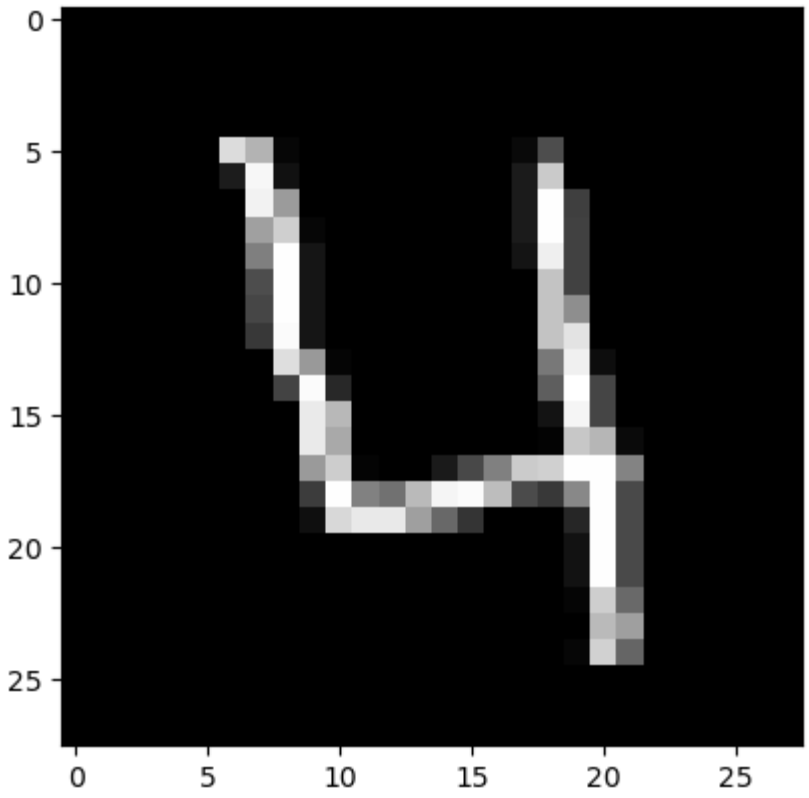
```
(784,)
```

In [26]:

```
four = four.values.reshape(28, 28)
plt.imshow(four, cmap='gray')
```

Out[26]:

```
<matplotlib.image.AxesImage at 0x1c5427380a0>
```



In [27]:

```
# visualise the array
print(four[5:-5, 5:-5])
```

```
[[  0 220 179   6   0   0   0   0   0   0   0   0   9  77   0   0   0   0]
 [  0  28 247  17   0   0   0   0   0   0   0   0  27 202   0   0   0   0]
 [  0   0 242 155   0   0   0   0   0   0   0   0  27 254  63   0   0   0]
 [  0   0 160 207   6   0   0   0   0   0   0   0  27 254  65   0   0   0]
 [  0   0 127 254  21   0   0   0   0   0   0   0  20 239  65   0   0   0]
 [  0   0  77 254  21   0   0   0   0   0   0   0   0 195  65   0   0   0]
 [  0   0  70 254  21   0   0   0   0   0   0   0   0 195 142   0   0   0]
 [  0   0  56 251  21   0   0   0   0   0   0   0   0 195 227   0   0   0]
 [  0   0   0 222 153   5   0   0   0   0   0   0   0 120 240  13   0   0]
 [  0   0   0  67 251  40   0   0   0   0   0   0   0  94 255  69   0   0]
 [  0   0   0   0 234 184   0   0   0   0   0   0   0  19 245  69   0   0]
 [  0   0   0   0 234 169   0   0   0   0   0   0   0   3 199 182  10   0]
 [  0   0   0   0 154 205   4   0   0  26  72 128 203 208 254 254 131   0]
 [  0   0   0   0  61 254 129 113 186 245 251 189  75  56 136 254  73   0]
 [  0   0   0   0  15 216 233 233 159 104  52   0   0   0  38 254  73   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  18 254  73   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  18 254  73   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   5 206 106   0]]
```

In [28]:

```
# Summarise the counts of 'label' to see how many labels of each digit are present
digits.label.value_counts()
```

Out[28]:

```
1    4684
7    4401
3    4351
9    4188
2    4177
6    4137
0    4132
4    4072
8    4063
5    3795
Name: label, dtype: int64
```

In [29]:

```
# missing values - there are none
digits.isnull().sum()
```

Out[29]:

```
label       0
pixel0      0
pixel1      0
pixel2      0
pixel3      0
           ..
pixel779    0
pixel780    0
pixel781    0
pixel782    0
pixel783    0
Length: 785, dtype: int64
```

In [30]:

```
# average values/distributions of features
description = digits.describe()
description
```

Out[30]:

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	
count	42000.000000	42000.0	42000.0	42000.0	42000.0	42000.0	42000.0	42000.0	42000.0	4
mean	4.456643	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
std	2.887730	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
min	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
25%	2.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
50%	4.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
75%	7.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
max	9.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

8 rows × 785 columns

Data Preparation for Model Building

In [31]:

```
# Creating training and test sets
# Splitting the data into train and test
X = digits.iloc[:, 1:]
Y = digits.iloc[:, 0]

# Rescaling the features
from sklearn.preprocessing import scale
X = scale(X)

# train test split with train_size=10% and test size=90%
x_train, x_test, y_train, y_test = train_test_split(X, Y, train_size=0.10, random_state=101
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(4200, 784)
(37800, 784)
(4200,)
(37800,)
```

Linear SVM

Let's first try building a linear SVM model (i.e. a linear kernel).

In [32]:

```
from sklearn import svm
from sklearn import metrics

# an initial SVM model with linear kernel
svm_linear = svm.SVC(kernel='linear')

# fit
svm_linear.fit(x_train, y_train)
```

Out[32]:

```
SVC(kernel='linear')
```

In [33]:

```
# predict
predictions = svm_linear.predict(x_test)
predictions[:10]
```

Out[33]:

```
array([1, 3, 0, 0, 1, 9, 1, 5, 0, 6], dtype=int64)
```

In [34]:

```
# evaluation: accuracy
# C(i, j) represents the number of points known to be in class i
# but predicted to be in class j
confusion = metrics.confusion_matrix(y_true = y_test, y_pred = predictions)
confusion
```

Out[34]:

```
array([[3615,    0,   12,    8,    8,   28,   28,    5,    9,    2],
       [   0, 4089,   16,   23,    9,    3,    3,   13,   25,    4],
       [  54,   48, 3363,   64,   74,   13,   53,   52,   59,   10],
       [  20,   28,  121, 3387,    8,  175,    5,   54,   58,   44],
       [  12,   12,   26,    2, 3399,    7,   41,   41,    4,  158],
       [  49,   42,   32,  177,   41, 2899,   54,   14,   82,   28],
       [  36,   16,   55,    5,   34,   37, 3486,    3,   21,    0],
       [   9,   27,   37,   22,   70,   10,    4, 3619,   14,  142],
       [  26,   86,   71,  137,   24,  137,   29,   26, 3096,   33],
       [  38,   11,   39,   26,  182,   19,    1,  207,   27, 3228]],
      dtype=int64)
```

In [35]:

```
# measure accuracy
metrics.accuracy_score(y_true=y_test, y_pred=predictions)
```

Out[35]:

```
0.9042592592592592
```

Non-Linear SVM

Let's now try a non-linear model with the RBF kernel.

In [36]:

```
# rbf kernel with other hyperparameters kept to default
svm_rbf = svm.SVC(kernel='rbf')
svm_rbf.fit(x_train, y_train)
```

Out[36]:

```
SVC()
```

In [37]:

```
# predict
predictions = svm_rbf.predict(x_test)

# accuracy
print(metrics.accuracy_score(y_true=y_test, y_pred=predictions))
```

```
0.9250793650793651
```

In [ ]: