In [8]:

```
import pandas as pd
import numpy as np
from math import sqrt
import time

import numba
from numba import int32, float64
from numba import njit, prange

from sklearn.model_selection import train_test_split
from sklearn.datasets import load_digits
from sklearn.metrics import accuracy_score

import matplotlib.pyplot as plt
##https://stackoverflow.com/questions/21154643/python-line-profiler-installa
%load_ext line_profiler
```

```
The line_profiler extension is already loaded. To reload it, use:
  %reload_ext line_profiler
```

In [9]:

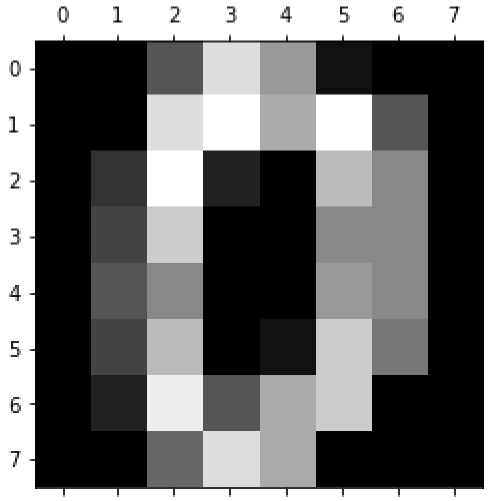
```
digits = load_digits()
print(digits.data.shape)
```

```
(1797, 64)
```

In [10]:

```
#image representatio of the data
plt.gray()
plt.matshow(digits.images[0])
plt.show()
```

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



In [11]:

```
X = digits.data
y = digits.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, ra
```

In [12]:
```
@numba.jit(nopython=True)
# @njit(parallel=True)
def euc_dist(x1, x2):
#     return np.sqrt(np.sum((x1-x2)**2
    dist = np.linalg.norm(x1-x2)
    return dist

@njit(parallel=True)
def calculate_all_dist(X_train,dist,test):
    for i in prange(X_train.shape[0]):
        dist[i] = euc_dist(X_train[i], test)
    return dist

@numba.jit(nopython=True)
def predict(X_train, y_train, test, K):
    dist = np.zeros((X_train.shape[0], 1))

    dist = calculate_all_dist(X_train,dist,test)
#     dist = np.array([euc_dist(test, x_t) for x_t in X_train])

    X_train = np.column_stack((X_train, y_train))
    X_train = np.column_stack((X_train, dist))

    X_train = X_train[X_train[:,-1].argsort()]

    neighbor_classes = X_train[:, -2][:K]
    classes = {}
    for item in neighbor_classes:
        if item in classes:
            classes[item] = classes.get(item) + 1
        else:
            classes[item] = 1
    counter_sorted = sorted(classes)

    return counter_sorted[0]

def predict_numba(X_train, X_test, y_train,K):
    predictions = np.zeros(X_test.shape[0])
    for i in np.arange(X_test.shape[0]):
        predictions[i] = predict(X_train, y_train, X_test[i], K)
    return predictions
```

In [14]:
```
k = 3
X_train, X_test, y_train = X_train.astype('float64'), X_test.astype('float64
start = time.time()
pred = predict_numba(X_train, X_test, y_train,k)
acc = accuracy_score(y_test, pred)
end = time.time()
print(f"Time Taken: {end-start} sec")
print("K = "+str(k)+"; Accuracy: "+str(acc))
```

```
Time Taken: 0.7539880275726318 sec
K = 3; Accuracy: 0.9711111111111111
```

In [23]:

```
%timeit -n 5 predict_numba(X_train, X_test, y_train,k)
```

```
378 ms ± 5.71 ms per loop (mean ± std. dev. of 7 runs, 5 loops each)
```

In [24]:

```
%lprun -f predict_numba predict_numba(X_train, X_test, y_train,k)
```

Logistic Regression

In [80]:

```
alpha = 1e-2

@numba.jit(nopython=True, parallel=True)
def logistic_regression(Y, X, w, iterations,alpha):
    for i in range(iterations):
            ypred =  _sigmoid(np.dot(X, w))
            gradient = np.dot((Y - ypred),X)
            w += np.dot(alpha, gradient)
    return w



def train(input_var, label, initial_params, iterations = 5000):
    """Train the model using batch gradient ascent"""
    x_total = []
    y_total = []
    for i, xy in enumerate(zip(input_var, label)):
                x_bar = np.array(np.concatenate((xy[0],[1.0]),axis=0))
                x_total.append(x_bar)
                y_binary = 1.0 if xy[1] == class_of_interest else 0.0
                y_total.append(y_binary)
    alphalist = np.ones(len(x_bar))*0.01
    gradient = logistic_regression(np.array(y_total),np.array(x_total),initi

    return gradient


def test(input_test, label_test,trained_params):
    """Test the accuracy of the model using test data"""
    total_classifications = 0
    correct_classifications = 0

    for x,y in zip(input_test, label_test):
        total_classifications += 1
        x_bar = np.array(np.concatenate((x,[1.0]),axis=0))
        y_hat = predict(x_bar, trained_params)

        y_binary = 1.0 if y == class_of_interest else 0.0

        if y_hat >= 0.5 and  y_binary == 1:
            correct_classifications += 1

        if y_hat < 0.5 and  y_binary != 1:
            correct_classifications += 1

    accuracy = correct_classifications / total_classifications

    return accuracy

```

In [81]:

```
digits_train, digits_test, digits_label_train, digits_label_test = train_tes
```

In [82]:

```
start = time.time()
initial_params = np.zeros(len(digits.data[0]) + 1)
for clas in range(10):
    class_of_interest = clas
    if clas == 0:
        trained_params = initial_params
    trained_params = train(digits_train / 16.0, digits_label_train, trained_
digits_accuracy = test(digits_test / 16.0, digits_label_test,trained_params)
end = time.time()

print(f'Accuracy of prediciting in test set: {digits_accuracy}')
print(f'Total time taken: {end- start} sec')
```

```
Accuracy of prediciting in test set: 0.9018518518518519
Total time taken: 1.2025914192199707 sec
```

In [83]:

```
%timeit -n 5  train(digits_train / 16.0, digits_label_train, initial_params,
```

```
884 ms ± 134 ms per loop (mean ± std. dev. of 7 runs, 5 loops each)
```

In [84]:

```
%lprun -f train train(digits_train / 16.0, digits_label_train, initial_param
```

In [ ]:

```

```