In [51]:

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

import numba
from numba import int32, float64
from numba.experimental import jitclass

from sklearn.model_selection import train_test_split
from sklearn.datasets import load_digits
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import accuracy_score, recall_score, precision_score
from sklearn.utils import shuffle


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

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

In [52]:

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


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


X = digits.data
y = digits.target

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

```
(1797, 64)
```

KNN

In [53]:

```
len(X_train),len(X_test)
```

Out[53]:

```
(1347, 450)
```

In [54]:

```
class KNN:
    def __init__(self, K=3):
        self.K = K

    def fit(self, x_train, y_train):
        self.X_train = x_train
        self.Y_train = y_train


    def euc_dist(self, x1, x2):
        return np.sqrt(np.sum((x1-x2)**2))


    def predict(self, X_test):
        predictions = []
        for i in range(len(X_test)):
            dist = np.array([self.euc_dist(X_test[i], x_t) for x_t in self.X
            dist_sorted = dist.argsort()[:self.K]
            neigh_count = {}
            for idx in dist_sorted:
                if self.Y_train[idx] in neigh_count:
                    neigh_count[self.Y_train[idx]] += 1
                else:
                    neigh_count[self.Y_train[idx]] = 1

            sorted_neigh_count = sorted(neigh_count.items(), reverse=True)
            predictions.append(sorted_neigh_count[0][0])
        return predictions

```

In [57]:

```
k = 3
start = time.time()
model = KNN(K = k)
model.fit(X_train, y_train)
pred = model.predict(X_test)
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: 13.793493747711182 sec
K = 3; Accuracy: 0.9755555555555555
```

In [58]:

```
%timeit -n 5 model.predict(X_test)
```

```
14.4 s ± 576 ms per loop (mean ± std. dev. of 7 runs, 5 loops each)
```

In [59]:

```
%lprun -f  model.predict model.predict(X_test)
```

Logistic regression

In [61]:

```python
alpha = 1e-2
class_of_interest = 10
max_iter = 1000


def _sigmoid(x):
    """Sigmoide function"""

    return 1.0 / (1.0 + np.exp(-x))

def predict(x_bar, params):
    """predict the probability of a class"""

    return _sigmoid(np.dot(params, x_bar))

def _compute_cost(input_var, output_var, params):
    """Compute the log likelihood cost"""

    cost = 0
    for x, y in zip(input_var, output_var):
        x_bar = np.array(np.insert(x, 0, 1))
        y_hat = self.predict(x_bar, params)

        y_binary = 1.0 if y == class_of_interest else 0.0
        cost += y_binary * np.log(y_hat) + (1.0 - y_binary) * np.log(1 - y_h

    return cost

def train(input_var, label, initial_params, print_iter = 5000):
    """Train the model using batch gradient ascent"""

    iteration = 1
    while iteration < max_iter:
        if iteration % print_iter == 0:
            print(f'iteration: {iteration}')
            print(f'cost: {_compute_cost(input_var, label, initial_params)}'
            print('--------------------------------------------')

        for i, xy in enumerate(zip(input_var, label)):
            x_bar = np.array(np.insert(xy[0], 0, 1))
            y_hat = predict(x_bar, initial_params)

            y_binary = 1.0 if xy[1] == class_of_interest else 0.0
            gradient = (y_binary - y_hat) * x_bar
            initial_params += alpha * gradient

        iteration +=1

    return initial_params

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.insert(x, 0, 1))
        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 classification of class_of_interest
            correct_classifications += 1

        if y_hat < 0.5 and  y_binary != 1:
            # correct classification of an other class
            correct_classifications += 1

    accuracy = correct_classifications / total_classifications

    return accuracy

```

In [62]:

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

In [63]:

```
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: 0.9851851851851852
Total time taken: 969.3034505844116sec
```

In [65]:

```
%timeit -n 1 train(digits_train / 16.0, digits_label_train, initial_params,1
```

```
1min 35s ± 10.8 s per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

In [66]:

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

In [ ]:

```

```