Machine Learning: Programming HW 1

Challa Priyankar — USN: `ENG20AM0017`

Raghav Nanjappan — USN: `ENG20AM0044`

Due Date: October 3^{rd}, 2022

1 Calculations

Our USNs are `ENG20AM0017` and `ENG20AM0044` respectively. As the required k- value is supposed to be the higher of the last two digits, the k- value for our classifier is 44.

For the number of data points n, we performed the following calculation:

$$(17 + 44) * 100 + 1000 = 7100$$

2 Code

This document has been typed out in LaTex, and since we are still learning how to use this, some comments and sources have been wrapped in more than 1 single comment. For a cleaner read of the code, please click on the following texts -

1. Custom Classifier
2. Library Classifer

and open the respective .ipynb file(s) on Jupyter Notebook or Google Colaboratory.

```
from keras.datasets import mnist
from sklearn.metrics import confusion_matrix as cm
from sklearn.utils import shuffle
import timeit as tt
import numpy as np
import pandas as pd

# Source: https://medium.com/analytics-vidhya/a-beginners-guide-to-knn
#-and-mnist-handwritten-digits-recognition-using-knn-from-scratch-df6fb982748a
```

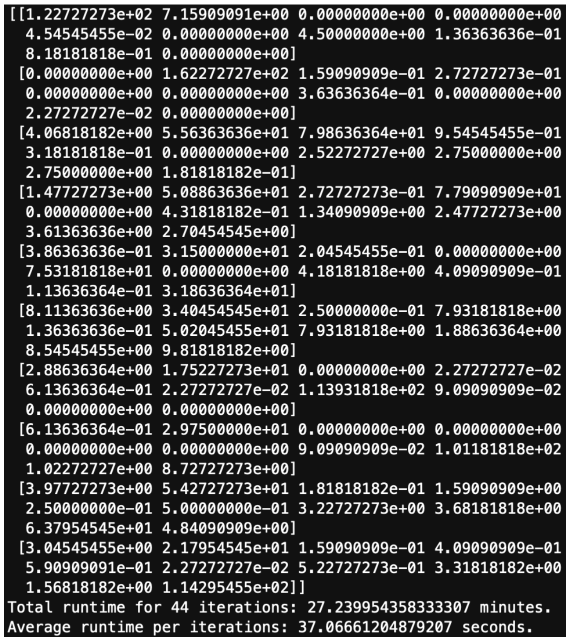
```
class kNN:
    def __init__(self, n_neighbors):
        self.K = n_neighbors
    def fit(self, train_X, train_y):
        self.train_X = train_X
        self.train_y = train_y
    def predict(self, test_X):
        predictions = []
        for i in range(len(test_X)):
            dist = np.array([eucDist(test_X[i], t_X) for t_X in self.train_X])
            distSort = dist.argsort()[:self.K]
            nCount = {} # Count of neighbors
            for idx in distSort:
                if self.train_y[idx] in nCount:
                    nCount[self.train_y[idx]] += 1
                else:
                    nCount[self.train_y[idx]] = 1
                sorted_nCount = sorted(nCount.items(), key=ig(1), reverse=True)
                predictions.append(sorted_nCount[0][0])
            return predictions
k = 44 # Since our USNs end with 17 and 44, the 'k' value is 44
dp = 7100 # Number of data points; (17 + 44) * 100 + 1000 = 7100
tp = 1420 # testing points; 20% of 7100 = 1420
n = 10 # Will be used for matrices and for-loops
(train_X, train_y), (test_X, test_y) = mnist.load_data()
train_X = train_X[:dp] # Resizing train_X with our required data points
train_X = np.reshape(train_X, (dp, 28 * 28))
train_y = train_y[:dp] # Resizing train_y with our required data points
test_X = test_X[:tp] # Resizing test_X with our required data points
test_X = np.reshape(test_X, (tp, 28 * 28))
test_y = test_y[:tp] # Resizing test_X with our required data points

# Custom shuffle methods since implementing mnist from keras in sklearn's train_test_split
# was complicated
# Source: Sai Nishwanth, USN: ENG21AM3031
def shufTrainData():
    X_shuf, y_shuf = shuffle(train_X, train_y)
    return (X_shuf, y_shuf)
def shufTestData():
    X_shuf, y_shuf = shuffle(test_X, test_y)
    return (X_shuf, y_shuf)
```

```
# Method to calculate the average runtime
def calcAvg(tAvg, total_1):
    if i == 0:
        tAvg = total_1
        return tAvg
    elif i != 0:
        tAvg = (tAvg + total_1)/(2)
        return tAvg
cm_0 = np.zeros((n, n)) # Creating a temporary null matrix

# To calculate the precision of each digit
# Source: Sai Nishwanth, USN: ENG21AM3031
prec = 0
# To calculate total runtime for k iterations
# Source: https://stackoverflow.com/questions/5622976/how-do-you-calculate-program-
# run-time-in-python
start_0 = tt.default_timer()
knn = kNN(n_neighbors = k)
tAvg = 0

for i in range(k):
    start_1 = tt.default_timer() # To calculate runtime for each iteration
    print(f'Starting iteration - {i}') # To keep a track of how many iterations have finished
    X_train, y_train = shufTrainData()
    X_test, y_test = shufTestData()
    X_train = X_train[:trp] # Training with 80% of data
    y_train = y_train[:trp]
    knn.fit(X_train, y_train)
    X_test = X_test[:tp] # Testing with 20% of data
    y_test = y_test[:tp]
    y_pred = knn.predict(X_test)
    cm_2 = cm(y_test, y_pred # Creating a confusion matrix to store summation
    # of all values after 'k' times
    cm_1 = cm_0 + cm_2
    cm_0 = cm_1
    prec += cm_0.diagonal()/cm_1.sum(axis = 0)
    print(f'Ending iteration - {i}')
    stop_1 = tt.default_timer() # To calculate runtime for each iteration
    total_1 = (stop_1 - start_1)
    tAvg = calcAvg(tAvg, total_1)
    print(f'Total runtime for iteration \'{i}\': {total_1} seconds.\n')
```

```
# To calculate total runtime for k iterations
stop_0 = tt.default_timer()
total_0 = (stop_0 - start_0)/60 # Since time is stored in seconds and it is easier to read in
# minutes
for i in range(n):
    print(f'Precision of number {i} - {(prec[i]/k)*100} %')
cmAvg = cm_1/k
print(cmAvg)
print(f'Total runtime for {k} iterations: {total_0} minutes.')
print(f'Average runtime per iterations: {tAvg} seconds.')
df = pd.DataFrame(cmAvg)
df
s = 0 # To calculate and verify the sum of all the elements in the matrix
for i in range (n):
    for j in range (n):
        s += cmAvg[i][j]
print(s)
```

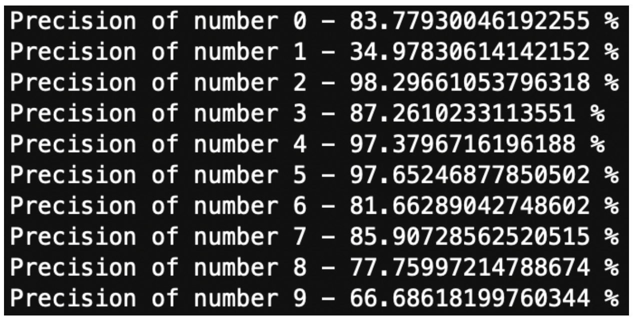
3 Confusion Matrices

3.1 Custom Classifier



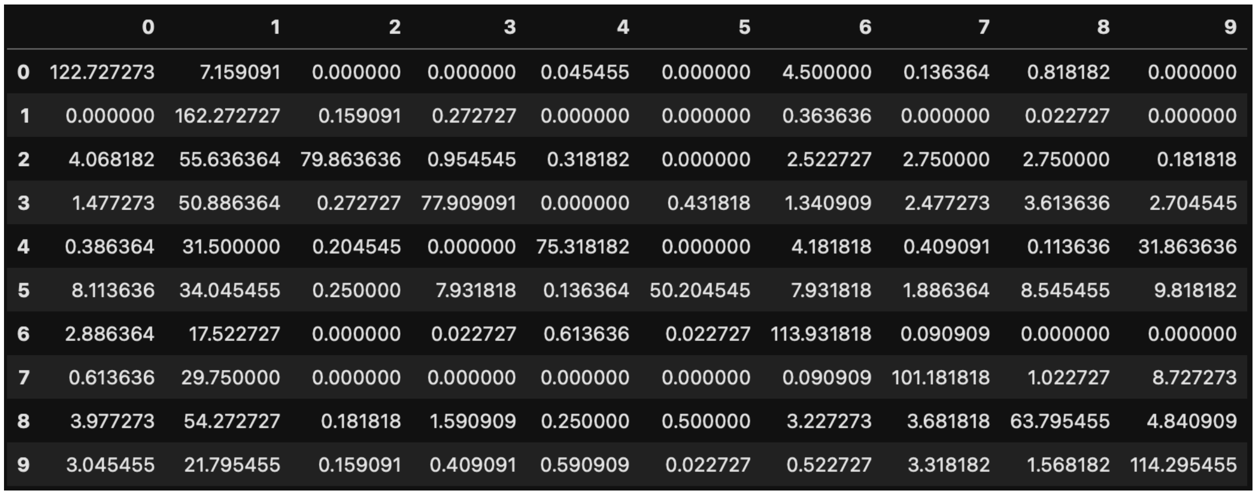
```
[[1.22727273e+02 7.15909091e+00 0.00000000e+00 0.00000000e+00
  4.54545455e-02 0.00000000e+00 4.50000000e+00 1.36363636e-01
  8.18181818e-01 0.00000000e+00]
 [0.00000000e+00 1.62272727e+02 1.59090909e-01 2.72727273e-01
  0.00000000e+00 0.00000000e+00 3.63636364e-01 0.00000000e+00
  2.27272727e-02 0.00000000e+00]
 [4.06818182e+00 5.56363636e+01 7.98636364e+01 9.54545455e-01
  3.18181818e-01 0.00000000e+00 2.52272727e+00 2.75000000e+00
  2.75000000e+00 1.81818182e-01]
 [1.47727273e+00 5.08863636e+01 2.72727273e-01 7.79090909e+01
  0.00000000e+00 4.31818182e-01 1.34090909e+00 2.47727273e+00
  3.61363636e+00 2.70454545e+00]
 [3.86363636e-01 3.15000000e+01 2.04545455e-01 0.00000000e+00
  7.53181818e+01 0.00000000e+00 4.18181818e+00 4.09090909e-01
  1.13636364e-01 3.18636364e+01]
 [8.11363636e+00 3.40454545e+01 2.50000000e-01 7.93181818e+00
  1.36363636e-01 5.02045455e+01 7.93181818e+00 1.88636364e+00
  8.54545455e+00 9.81818182e+00]
 [2.88636364e+00 1.75227273e+01 0.00000000e+00 2.27272727e-02
  6.13636364e-01 2.27272727e-02 1.13931818e+02 9.09090909e-02
  0.00000000e+00 0.00000000e+00]
 [6.13636364e-01 2.97500000e+01 0.00000000e+00 0.00000000e+00
  0.00000000e+00 0.00000000e+00 9.09090909e-02 1.01181818e+02
  1.02272727e+00 8.72727273e+00]
 [3.97727273e+00 5.42727273e+01 1.81818182e-01 1.59090909e+00
  2.50000000e-01 5.00000000e-01 3.22727273e+00 3.68181818e+00
  6.37954545e+01 4.84090909e+00]
 [3.04545455e+00 2.17954545e+01 1.59090909e-01 4.09090909e-01
  5.90909091e-01 2.27272727e-02 5.22727273e-01 3.31818182e+00
  1.56818182e+00 1.14295455e+02]]
Total runtime for 44 iterations: 27.239954358333307 minutes.
Average runtime per iterations: 37.06661204879207 seconds.
```

Figure 3.1.1: Averaged Confusion Matrix with runtime using Custom Classifier



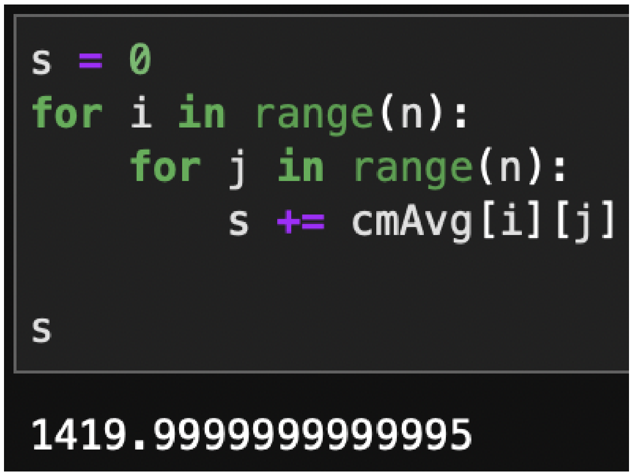
```
Precision of number 0 - 83.77930046192255 %
Precision of number 1 - 34.97830614142152 %
Precision of number 2 - 98.29661053796318 %
Precision of number 3 - 87.2610233113551 %
Precision of number 4 - 97.3796716196188 %
Precision of number 5 - 97.65246877850502 %
Precision of number 6 - 81.66289042748602 %
Precision of number 7 - 85.90728562520515 %
Precision of number 8 - 77.75997214788674 %
Precision of number 9 - 66.68618199760344 %
```

Figure 3.1.2: Precision of kNN algorithm using Custom Classifier



	0	1	2	3	4	5	6	7	8	9
0	122.727273	7.159091	0.000000	0.000000	0.045455	0.000000	4.500000	0.136364	0.818182	0.000000
1	0.000000	162.272727	0.159091	0.272727	0.000000	0.000000	0.363636	0.000000	0.022727	0.000000
2	4.068182	55.636364	79.863636	0.954545	0.318182	0.000000	2.522727	2.750000	2.750000	0.181818
3	1.477273	50.886364	0.272727	77.909091	0.000000	0.431818	1.340909	2.477273	3.613636	2.704545
4	0.386364	31.500000	0.204545	0.000000	75.318182	0.000000	4.181818	0.409091	0.113636	31.863636
5	8.113636	34.045455	0.250000	7.931818	0.136364	50.204545	7.931818	1.886364	8.545455	9.818182
6	2.886364	17.522727	0.000000	0.022727	0.613636	0.022727	113.931818	0.090909	0.000000	0.000000
7	0.613636	29.750000	0.000000	0.000000	0.000000	0.000000	0.090909	101.181818	1.022727	8.727273
8	3.977273	54.272727	0.181818	1.590909	0.250000	0.500000	3.227273	3.681818	63.795455	4.840909
9	3.045455	21.795455	0.159091	0.409091	0.590909	0.022727	0.522727	3.318182	1.568182	114.295455

Figure 3.1.3: Averaged Confusion Matrix in Pandas Dataframe using Custom Classifier

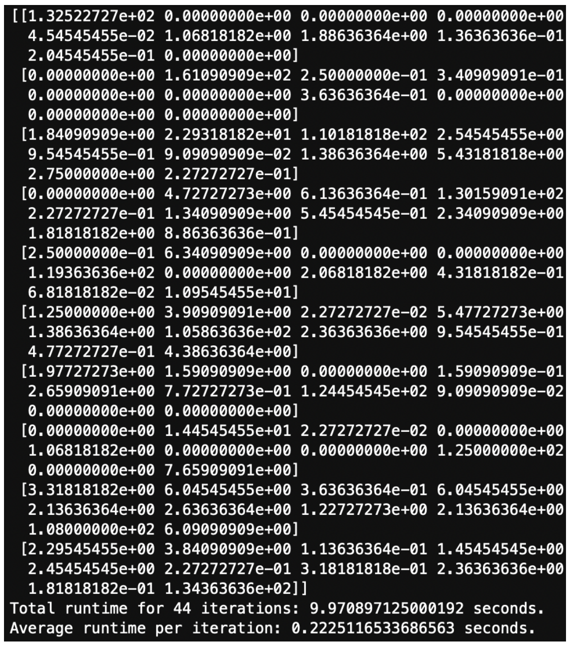


```
s = 0
for i in range(n):
    for j in range(n):
        s += cmAvg[i][j]


s
```

```
1419.9999999999995
```

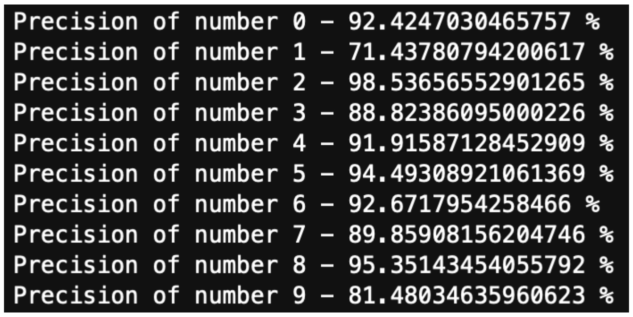
Figure 3.1.4: Verification of total number of testing points using Custom Classifier

3.2 Library Classifier



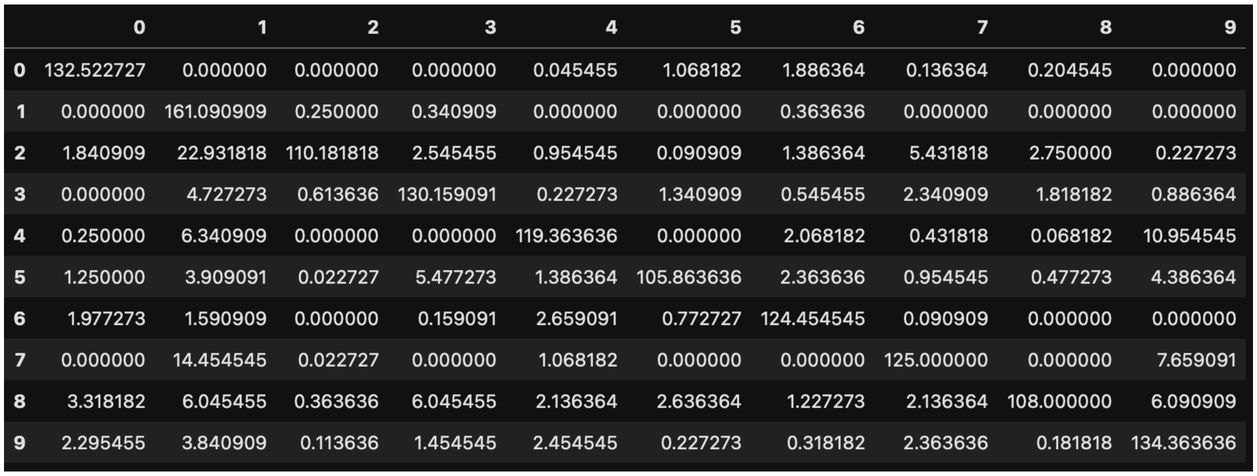
```
[[1.32522727e+02 0.00000000e+00 0.00000000e+00 0.00000000e+00
  4.54545455e-02 1.06818182e+00 1.88636364e+00 1.36363636e-01
  2.04545455e-01 0.00000000e+00]
 [0.00000000e+00 1.61090909e+02 2.50000000e-01 3.40909091e-01
  0.00000000e+00 0.00000000e+00 3.63636364e-01 0.00000000e+00
  0.00000000e+00 0.00000000e+00]
 [1.84090909e+00 2.29318182e+01 1.10181818e+02 2.54545455e+00
  9.54545455e-01 9.09090909e-02 1.38636364e+00 5.43181818e+00
  2.75000000e+00 2.27272727e-01]
 [0.00000000e+00 4.72727273e+00 6.13636364e-01 1.30159091e+02
  2.27272727e-01 1.34090909e+00 5.45454545e-01 2.34090909e+00
  1.81818182e+00 8.86363636e-01]
 [2.50000000e-01 6.34090909e+00 0.00000000e+00 0.00000000e+00
  1.19363636e+02 0.00000000e+00 2.06818182e+00 4.31818182e-01
  6.81818182e-02 1.09545455e+01]
 [1.25000000e+00 3.90909091e+00 2.27272727e-02 5.47727273e+00
  1.38636364e+00 1.05863636e+02 2.36363636e+00 9.54545455e-01
  4.77272727e-01 4.38636364e+00]
 [1.97727273e+00 1.59090909e+00 0.00000000e+00 1.59090909e-01
  2.65909091e+00 7.72727273e-01 1.24454545e+02 9.09090909e-02
  0.00000000e+00 0.00000000e+00]
 [0.00000000e+00 1.44545455e+01 2.27272727e-02 0.00000000e+00
  1.06818182e+00 0.00000000e+00 0.00000000e+00 1.25000000e+02
  0.00000000e+00 7.65909091e+00]
 [3.31818182e+00 6.04545455e+00 3.63636364e-01 6.04545455e+00
  2.13636364e+00 2.63636364e+00 1.22727273e+00 2.13636364e+00
  1.08000000e+02 6.09090909e+00]
 [2.29545455e+00 3.84090909e+00 1.13636364e-01 1.45454545e+00
  2.45454545e+00 2.27272727e-01 3.18181818e-01 2.36363636e+00
  1.81818182e-01 1.34363636e+02]]
Total runtime for 44 iterations: 9.970897125000192 seconds.
Average runtime per iteration: 0.2225116533686563 seconds.
```

Figure 3.2.1: Averaged Confusion Matrix with runtime using Library Classifier



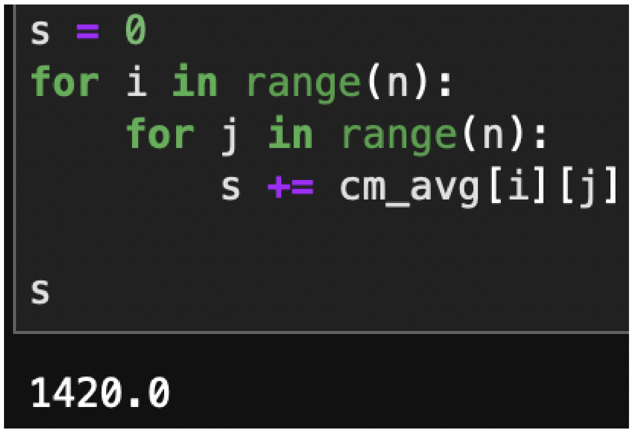
```
Precision of number 0 - 92.4247030465757 %
Precision of number 1 - 71.43780794200617 %
Precision of number 2 - 98.53656552901265 %
Precision of number 3 - 88.82386095000226 %
Precision of number 4 - 91.91587128452909 %
Precision of number 5 - 94.49308921061369 %
Precision of number 6 - 92.6717954258466 %
Precision of number 7 - 89.85908156204746 %
Precision of number 8 - 95.35143454055792 %
Precision of number 9 - 81.48034635960623 %
```

Figure 3.2.2: Precision of kNN algorithm using Library Classifier



	0	1	2	3	4	5	6	7	8	9
0	132.522727	0.000000	0.000000	0.000000	0.045455	1.068182	1.886364	0.136364	0.204545	0.000000
1	0.000000	161.090909	0.250000	0.340909	0.000000	0.000000	0.363636	0.000000	0.000000	0.000000
2	1.840909	22.931818	110.181818	2.545455	0.954545	0.090909	1.386364	5.431818	2.750000	0.227273
3	0.000000	4.727273	0.613636	130.159091	0.227273	1.340909	0.545455	2.340909	1.818182	0.886364
4	0.250000	6.340909	0.000000	0.000000	119.363636	0.000000	2.068182	0.431818	0.068182	10.954545
5	1.250000	3.909091	0.022727	5.477273	1.386364	105.863636	2.363636	0.954545	0.477273	4.386364
6	1.977273	1.590909	0.000000	0.159091	2.659091	0.772727	124.454545	0.090909	0.000000	0.000000
7	0.000000	14.454545	0.022727	0.000000	1.068182	0.000000	0.000000	125.000000	0.000000	7.659091
8	3.318182	6.045455	0.363636	6.045455	2.136364	2.636364	1.227273	2.136364	108.000000	6.090909
9	2.295455	3.840909	0.113636	1.454545	2.454545	0.227273	0.318182	2.363636	0.181818	134.363636

Figure 3.2.3: Averaged Confusion Matrix in Pandas Dataframe using Library Classifier



```
s = 0
for i in range(n):
    for j in range(n):
        s += cm_avg[i][j]

s
```

```
1420.0
```

Figure 3.2.4: Verification of total number of testing points using Library Classifier

4 General Observations and Inferences

1. As we are taking the average value of k confusion matrices, the elements in the final averaged confusion matrix all seem have decimal places as they don't necessarily need to be divisible by k.

2. In Figure 3.1.4, the sum is not exactly the value of n. This could be because we are taking the averaged value confusion matrix and due to this, there might be some losses in the decimal places.

3. The time difference between the library classifier($\sim$10 seconds, see Figure 3.2.1) and the custom algorithm($\sim$27 minutes, see Figure 3.1.1) is very high. Our assumption is that calculating the Euclidean distance between a particular point in the dataset with every other point in the dataset is what is taking too long.