Homework #4: Multicore Programming
Dayanara Lebron Aldea
STA 141C: High Performance Computing

Task: Compare your run-time with the original one.

I used my personal computer to run this, below is a screenshot of my run-time and accuracy and the original single threaded and its accuracy. The times achieved are machine-dependent, in my computer the original script took ~199s to run compared to 79.8s with the 4-core paralleled code. This represents a 60% decrease in running time, making it about 2.5 times faster. Accuracies remained the same.

```
Dayanaras-MacBook-Pro:hw4_code 2 dlaldea$ python go_knn_edit.py
Accuracy 0.794000 Time 79.793315 secs.

Dayanaras-MacBook-Pro:hw4_code 2 dlaldea$ python go_knn.py
Accuracy 0.794000 Time 198.576497 secs.
```

Snippet of Code:

Since the knn is done for each testing sample, and we needed to divide the calculations in 4 core processes; I divided the testing data in 4 subsets, making each set a parallel job. Each job calculates the amount of correct predictions and at the end we sum all correct predictions across processes and compute the accuracy. Below how I parallelized the code:

```
def go_nn(Xtrain, ytrain, Xtest, ytest):


  def knn_per_proc(start, finish,output):
    correct =0
    for i in range(start,finish): ## For all testing instances
      nowXtest = Xtest[i,:]
      ### Find the index of nearest neighbor in training data
      dis_smallest = np.linalg.norm(Xtrain[0,:]-nowXtest)
      idx = 0
      for j in range(1, Xtrain.shape[0]):
        dis = np.linalg.norm(nowXtest-Xtrain[j,:])
        if dis < dis_smallest:
          dis_smallest = dis
          idx = j
    ### Now idx is the index for the nearest neighbor
```

```
    ## check whether the predicted label matches the true label
        if ytest[i] == ytrain[idx]:
            correct += 1
    output.put(correct)

#Parallelize
n_processes=4
n_items=Xtest.shape[0]

#Calculate number of items per process
i_pp=(n_items+n_processes-1)/n_processes
output=mp.Queue()

#Calculate starting and stopping points in data
stop=[(i,min(n_items,i+i_pp)) for i in range(0,n_items,i_pp)]

 #Create processes
processes=[mp.Process(target=knn_per_proc, args=(t[0],t[1],output)) for t in stop]

#Run processes
for p in processes:
    p.start()

#Exit the completed processes
for p in processes:
    p.join()

# sum of correctly predicted per process
 correct=sum([output.get() for p in processes])
 acc = correct/float(Xtest.shape[0])
 return acc
```