Katherine Olson

STA 141C Homework #4

Problem:

From http://www.stat.ucdavis.edu/~chohsieh/teaching/STA141C_Spring2017/hw4_code.zip, use that data and code to parallelize the computation using multiple cores. Compare the two versions.

Code:

I modified the go_nn function to parallelize the original function by using four cores. It now calls another function for each fourth of the testing data. Instead of looping through each row of the testing data to find the accuracy, it loops through each fourth of the data at the same time.

```
def go_nn(Xtrain, ytrain, Xtest, ytest):
        """
        Input: Testing and training data.
        Output: The number of correct predictions when using k-nearest neighbors on that data.
        """

        size = Xtest.shape[0] / 4 # How many rows are 1/4 of the data
        corr = 0

        # Setup a list of processes
        plist = []
        q = mp.Queue() # Store number correct for each core

        # Creates the processes:
        for i in range(1, 5):
                plist.append(mp.Process(target = eachSection, args=(Xtrain, ytrain, Xtest, ytest,
                                                        range((size * i) - size, size * i), q)))

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

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

        # Get the total number of correct predictions:
        while (q.empty() == False):
                corr += q.get()

        acc = corr/float(Xtest.shape[0])
        return acc
```

Each fourth of the data is sent to the eachSection function below. This function finds the number of correct predictions for that chunk of data.

```
def eachSection(Xtrain, ytrain, Xtest, ytest, indices, q):
        """
        Input: The testing and training data, the indices for the core to use, and the queue storing
the results.
        Output: Inserts the number of correct predictions for that chunck of testing data.
        """
        correct =0
        for i in indices: ## For all testing instances in this core (1/4)
                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

        # Add to the queue:
        q.put(correct)
        return
```

Results:

	Accuracy	**Time**
1 Core	0.794	156.900 seconds
4 Cores	0.794	69.331 seconds

So, as expected, the program that used four cores was faster. The accuracy is the same, which means the two programs do the same thing, as they should.