```
import os
import struct
from mnist import read, show
import numpy as np
from collections import Counter
import time

start_time = time.time()
```

read MNIST database info

reading functions from: https://gist.github.com/akesling/5358964

```
training_data = list(read(dataset = "training", path = "./data"))
testing_data = list(read(dataset = "testing", path = "./data"))
```

Find distance between trainDigit and testDigit

Pixel-wise absolute value difference

```
def calcDistL1(testval, trainval):
dist = np.sum(np.abs(testval-trainval))
return dist

#
def nearestNeighbor(testval):
testlabel, testpixels = testval
min_dist = 10000000
distances = []
for i in xrange(len(training_data)):
```

tlabel, tpixels = training_data[i]
dist = calcDistL1(testpixels, tpixels)
data = (dist, tlabel)
distances.append(data)
distances.sort()
return distances

def knn(k, testDigitIndex):
# show(testing_data[testDigitIndex][1])
label = testing_data[testDigitIndex][0]
print("Test Digit Label: " + str(label))
sorted_dists = nearestNeighbor(testing_data[testDigitIndex])
knnObjs = sorted_dists[:k]

```
# strip list to be list of nearest labels
nearest = []
for x in xrange(len(knnObjs)):
    t, l = knnObjs[x]
    nearest.append(l)

this_data = Counter(nearest)
vote = this_data.most_common(1)
prediction = vote[0][0]
return (label, prediction)
```

Inputs: (testing_length, k, fileout_name, dist_type)

Training length: How much of the testing set should it run the prediction on

k: Pass in Value of k

fileout_name: Name of file to write results to

dist_type: Which distance formula to use: 1: L1, or 2: L2

def mainKNN(testing_length, k):
totalcorrect = 0
for i in xrange(testing_length):
print(str(i+1) + " out of " + str(testing_length))
result = knn(5, i)
l, p = result
print("Predicted label: " + str(p))
if l == p:
print("CORRECT")
totalcorrect = totalcorrect + 1.0
else:
print("INCORRECT")
print("\n")
# show(testing_data[i][1]) # uncomment to view digit in numpy graph

```
print("Total correct: " + str(totalcorrect))
accuracy = float(totalcorrect / testing_length)*100.00
print("Accuracy: " + str(accuracy) + "%")

print("--- %s seconds ---" % (time.time() - start_time))
```

mainKNN(1000, 5)