```
# make a label dictionary with the key as labels and list of centroid classes as
    the values
import random
import math
import sys

MAX = 100

class Datapoint(object):
  def __init__(self, vector):
    self.vector = vector
    self.label = None
  def set_label(self, label):
    self.label = label

def readFile():
  dataSet = list()
  fp = open('mnist_data.txt', 'r').readlines()
  kp = open('mnist_labels.txt', 'r').readlines()
  if len(fp) != len(kp):
    print "Not as many labels in the dataset as there are data points"
  else:
    for index, ii in enumerate(fp):
      dp = Datapoint([int(el) for el in ii.strip().split(" ")])
      dp.set_label(kp[index].strip())
      dataSet.append(dp)
    return dataSet

def getRandomCentroids(dataset, k):
  return [dataset[random.randint(0, len(dataset))] for ii in range(k)]

def getDistinctCentroids(dataset, k):
  label = 0
  distinct_label_centroids = list()
  for ii in dataset:
    if ii.label == str(label):
      distinct_label_centroids.append(ii)
      label += 1
  return distinct_label_centroids


def calc_euclid_distance(vector_one, vector_two):
  sum = 0
  for ii in range(0, 784):
    sum += ((vector_one[ii] - vector_two[ii]) ** 2)
  return math.sqrt(sum)

def kmeans(dataset, k, distinct = False):
  #Dataset in this case is a list of centroid class
  #Return a random set of centroid class. The set should be of size k
  if not distinct:
    centroids = getRandomCentroids(dataset, k)
  else:
    centroids = getDistinctCentroids(dataset, k)
  #Inititalize a book keeping variable
  iterations = 0
```

```
  oldCentroids = None
  while not ShouldStop(oldCentroids, centroids, iterations):
    old_centroids = centroids
    iterations += 1
    #assign to differenct centroid classes
    assignments = get_assignments(dataset, centroids)
    centroids = getCentroids(dataset, assignments)
  return assignments

def calculate_average(col_vectors):
  vector = list()
  for index in range(0, 784):
    sum = 0
    for vectorval in col_vectors:
      sum += vectorval.vector[index]
    sum /= len(col_vectors)
    vector.append(sum)
  return vector

def getCentroids(dataset, assignments):
  centroids = list()
  for value in assignments.values():
    centroids.append(Datapoint(calculate_average(value)))
  return centroids

def ShouldStop(oldCentroids, centroids, iterations):
  global MAX
  if iterations > MAX:
    return True
  if oldCentroids == None:
    return False
  else:
    return [ii.vector for ii in centroids] == [jj.vector for jj in oldCentroids]

def get_assignments(dataset, centroids):
  assignments = {}
  for ii in dataset:
    closest_centroid = None
    old_distance = float("inf")
    for centroid in centroids:
      distance = calc_euclid_distance(centroid.vector, ii.vector)
      if distance < old_distance or closest_centroid == None:
        old_distance = distance
        closest_centroid = centroid
    try:
      assignments[closest_centroid].append(ii)
    except:
      assignments[closest_centroid] = [ii]
  return assignments

def calculate_maps(assignments, fp, iteration):
  centroid_labels = []
  for ii in assignments.values():
    class_label_list = list(set([elem.label for elem in ii]))
    max_num = 0
    max_label = None
```

```
      for label in class_label_list:
        label_num = len(filter(lambda xx: xx.label == label, ii))
        if label_num > max_num:
          max_num = label_num
          max_label  = label
      centroid_labels.append((max_label, max_num))
    fp.write("=============================================================\n")
    fp.write(" - - - - - - - This is Iteration no %s --------------------\n" %
        iteration)
    fp.write("=============================================================\n")
    for index, ii in enumerate(centroid_labels):
      fp.write("The label for the %s centroid is %s with %s elements\n" %(index + 1
          , ii[0],ii[1]))
      fp.write("- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -\n")

  def main():
    dataset = readFile()
    fp = open("outputs_10.txt", "w")
    for ii in range(1, 11):
      #pass parameters to the kmeans function as per your need.
      assignments = kmeans(dataset, 10)
      calculate_maps(assignments, fp, ii)

  main()
```