K-means Sample Code

import numpy as np

# Returns an array of centers chosen at random from data

def get\_rand\_centers(data, num\_centers):

used = set()

centers = np.empty(num\_centers)

for i in range(0, num\_centers):

t = random.randint(0, len(data)-1)

while t in used:

t = random.randint(0, len(data)-1)

centers[i] = data[t]

used.add(t)

return centers

# Returns the 'distance' between two values

def dist(x,y):

return (x-y)\*\*2

# Returns an updated centers array

def update\_centers(centers, clusters, data):

num\_centers = len(centers)

data\_size = len(data)

temp = np.zeros(num\_centers)

for i in range(0, data\_size):

temp[int(clusters[i])] += data[i]

uniq, counts = np.unique(clusters, return\_counts=True)

centers = np.true\_divide(temp,counts)

return centers

# Returns an array with the cluster each data point belongs to

# Checks for empty clusters and reassigned at random from data

def classify(centers, data):

data\_size = len(data)

cluster\_num = len(centers)

clusters = np.empty(data\_size)

temp = np.empty(cluster\_num)

for i in range(0, data\_size):

for j in range(0, cluster\_num):

temp[j] = dist(centers[j],data[i])

clusters[i] = np.argmin(temp)

uniq, counts = np.unique(clusters, return\_counts=True)

while(len(uniq) != len(centers)):

reassign = set(range(0,len(centers))) - set(uniq)

for i in reassign:

centers[i] = data[random.randint(0,len(data)-1)]

clusters = classify(centers, data)

uniq, counts = np.unique(clusters, return\_counts=True)

return clusters

# The k-means algorithm

# Returns an array of centers, and an array of associated variances

def k\_means(data, num\_centers):

centers = get\_rand\_centers(data, num\_centers)

old\_centers = None

iterations = 0

while(not np.array\_equal(old\_centers, centers)):

old\_centers = np.copy(centers)

iterations += 1

clusters = classify(centers, data)

centers = update\_centers(centers, clusters, data)

return centers