Assignment04

April 4, 2019

Assignment04: K-means clustering
Software Engineering
20154652 Lee Dong Jae

```
In [1]: import matplotlib.pyplot as plt
        import numpy as np

        file_data                = "mnist_train.csv"
        test_data = "mnist_test.csv"

        handle_file        = open(file_data, "r")
        data                        = handle_file.readlines()
        handle_file.close()

        handle_file = open(test_data, "r")
        test = handle_file.readlines()
        handle_file.close()

        size_row        = 28    # height of the image
        size_col          = 28    # width of the image

        num_image        = len(data)
        test_num_image = len(test)
        count               = 0     # count for the number of images|
        test_count = 0
```

normalize the values of the input data to be [0, 1]

```
In [2]: def normalize(data):

            data_normalized = (data - min(data)) / (max(data) - min(data))

            return(data_normalized)
```

example of distance function between two vectors x and y

```
In [3]: def distance(x, y):
```

```
        d = (x - y) ** 2
        #s = np.sum(d)
        #r = np.sqrt(s)

        return(d)
```

make a matrix each column of which represents an images in a vector form

```
In [4]: list_image  = np.empty((size_row * size_col, num_image), dtype=float)
        list_label  = np.empty(num_image, dtype=int)
        test_list_image  = np.empty((size_row * size_col, test_num_image), dtype=float)
        test_list_label  = np.empty(test_num_image, dtype=int)
```

```
In [5]: for line in data:

            line_data   = line.split(',')
            label       = line_data[0]
            im_vector   = np.asfarray(line_data[1:])
            im_vector   = normalize(im_vector)

            list_label[count]       = label
            list_image[:, count]    = im_vector

            count += 1
```

```
In [6]: for line in test:

            line_data   = line.split(',')
            label       = line_data[0]
            im_vector   = np.asfarray(line_data[1:])
            im_vector   = normalize(im_vector)

            test_list_label[test_count]       = label
            test_list_image[:, test_count]    = im_vector

            test_count += 1
```

```
In [7]: real_list_image = list_image.T
        real_test_list_image = test_list_image.T
```

Functions for obtaining mode value

```
In [8]: from collections import Counter

        def mode(numbers):
            c = Counter(numbers)
            mode = c.most_common(1)

            return mode[0][0]
```

Initialize the centroids randomly

```
In [9]: def initialize_centroid(k):

            #Assign a dictionary that contain energy, training accuracy, test accuracy
            info = {'energy_list': [], 'training_acc': [], 'test_acc': []}

            first_centroid = np.zeros((k, 784))
            initial_label_idx = np.random.randint(0, k, size=60000)
            for i in range(k):
                temp_idx_list = list(np.where(initial_label_idx == i)[0])
                for j in temp_idx_list:
                    first_centroid[i] += real_list_image[j]

                first_centroid[i] /= len(temp_idx_list)

            do_clustering(first_centroid, k, info)
```

Function to obtain accuracy

```
In [10]: def training_find_accuracy(clusters, k):
             ans_count = 0
             for i in range(k):
                 temp_label = []
                 for j in clusters[i]:
                     temp_label.append(list_label[j])
                 mod = mode(temp_label)
                 ans_count += temp_label.count(mod)

             accuracy = ans_count/60000
             return accuracy

In [11]: def test_find_accuracy(clusters, k):
             ans_count = 0
             for i in range(k):
                 temp_label = []
                 for j in clusters[i]:
                     temp_label.append(test_list_label[j])
                 mod = mode(temp_label)
                 ans_count += temp_label.count(mod)

             accuracy = ans_count/10000
             return accuracy
```

Function to obtain energy

```
In [12]: def energy_function(centroid, clusters, k):
             energy = 0
             for i in range(k):
```

```
            for j in clusters[i]:
                energy += np.sum(distance(real_list_image[j], centroid[i]))

        return energy / 60000
```

Function for clustering

```
In [13]: def do_clustering(centroid, k, info):

            #make a place for put indexes of data to each clusters
            clusters = {idx: [] for idx in range(0, k)}
            test_clusters = {idx: [] for idx in range(0, k)}

            #a temporary array for keeping the distance
            temp_distance = np.empty(k)

            for i in range(60000):
                for j in range(k):
                    temp_distance[j] = np.sum(distance(real_list_image[i], centroid[j]))
                #find the argmin of distance. And append a idx of data to the cluster[argmin]
                clusters[np.argmin(temp_distance)].append(i)

            #the same as above
            for m in range(10000):
                for n in range(k):
                    temp_distance[n] = np.sum(distance(real_test_list_image[m], centroid[n]))
                test_clusters[np.argmin(temp_distance)].append(m)

            #calcuate energy, training accuracy, test accuracy at each time
            energy2 = energy_function(centroid, clusters, k)
            training_accuracy = training_find_accuracy(clusters, k)
            test_accuracy = test_find_accuracy(test_clusters, k)

            print(energy2, ' / ', training_accuracy, ' / ', test_accuracy)

            #append calculated information to each list
            info['energy_list'].append(energy2)
            info['training_acc'].append(training_accuracy)
            info['test_acc'].append(test_accuracy)

            make_new_centroid(centroid, clusters, k, info)
```

Function to obtain a new centroids

```
In [14]: def make_new_centroid(centroid, clusters, k, info):

            new_centroid = np.zeros((k, 784))

            #sum previous data contained in same cluster
```

```
            for i in range(k):
                for j in clusters[i]:
                    new_centroid[i] += real_list_image[j]

            for m in range(k):
                new_centroid[m] = new_centroid[m] / len(clusters[m])

            #if clustering does not change over, plot images and information
            if np.array_equal(centroid, new_centroid):
                print('end')
                plot_image(centroid, clusters, k)
                plot_charts(info)
            else:
                do_clustering(new_centroid, k, info)
```

Visualize K centroid images for each category

```
In [15]: def plot_image(centroid, clusters, k):
             plt.figure(1)
             for idx, value in enumerate(centroid):
                 im_matrix = value.reshape((size_row, size_col))
                 plt.subplot(1, k, idx + 1)
                 plt.title(f"#{idx}")
                 plt.imshow(im_matrix, cmap='Greys', interpolation='None')
                 frame = plt.gca()
                 frame.axes.get_xaxis().set_visible(False)
                 frame.axes.get_yaxis().set_visible(False)
             plt.show()
```

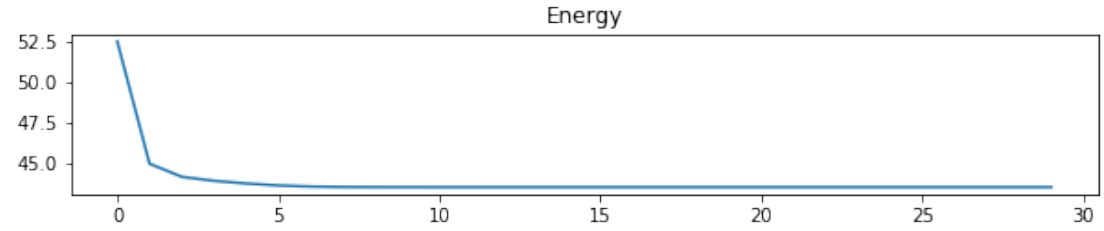
Plot the training energy/ training accuracy/ testing accuracy per optimization iteration.

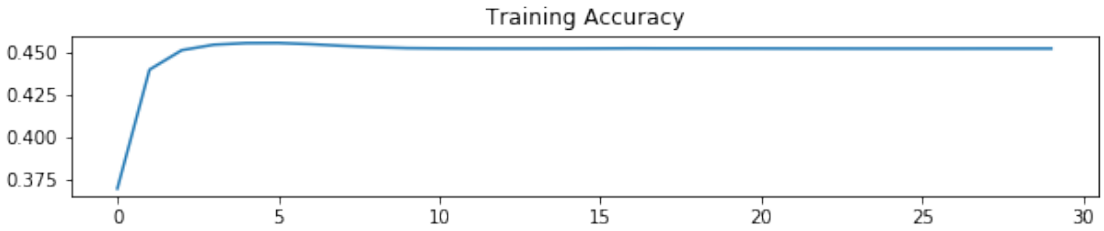
```
In [16]: def plot_charts(info):

             plt.figure(figsize=(10, 8))
             plt.tight_layout()

             plt.subplots_adjust(hspace = 1)

             plt.subplot(311)
             plt.title("Energy")
             plt.plot(range(len(info['energy_list'])), info['energy_list'])

             plt.subplot(312)
             plt.title("Training Accuracy")
             plt.plot(range(len(info['training_acc'])), info['training_acc'])

             plt.subplot(313)
             plt.title("Test Accuracy")
```

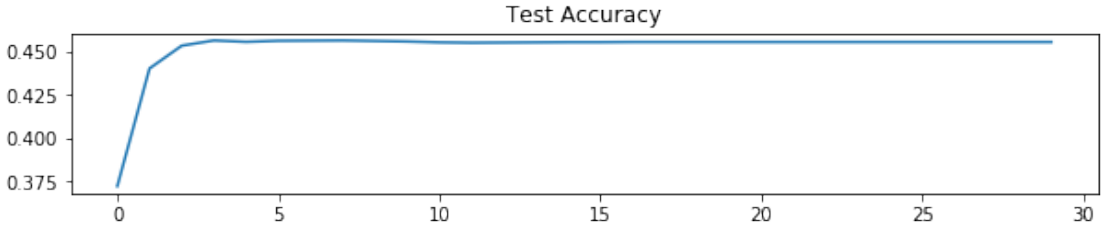
```
            plt.plot(range(len(info['test_acc'])), info['test_acc'])

            plt.show()
```

```
In [17]: initialize_centroid(5)
```

```
52.53708658974705  /  0.36956666666666665  /  0.3725
44.92948289774081  /  0.43983333333333335  /  0.4401
44.11163906559478  /  0.45131666666666664  /  0.4532
43.86284817941467  /  0.45458333333333334  /  0.4562
43.695644809328  /  0.4555166666666667  /  0.4554
43.57414275804744  /  0.45558333333333334  /  0.456
43.507570965619266  /  0.4548833333333333  /  0.4561
43.47962882037572  /  0.45386666666666664  /  0.4562
43.47134318422009  /  0.45308333333333334  /  0.4559
43.46884089561975  /  0.45253333333333334  /  0.4556
43.467757923838086  /  0.45238333333333336  /  0.4551
43.46721083772308  /  0.4522833333333333  /  0.4549
43.46691468731592  /  0.45226666666666665  /  0.455
43.46673446400926  /  0.45226666666666665  /  0.4551
43.46662744020971  /  0.4522833333333333  /  0.4552
43.46658174261212  /  0.45235  /  0.4552
43.46655297278544  /  0.45243333333333335  /  0.4553
43.46653742301138  /  0.45238333333333336  /  0.4553
43.46652927509492  /  0.45238333333333336  /  0.4553
43.46652551008315  /  0.45236666666666664  /  0.4553
43.46652369024622  /  0.45235  /  0.4553
43.46652185612527  /  0.45231666666666664  /  0.4553
43.4665205813426  /  0.4523  /  0.4553
43.46651905750297  /  0.4522833333333333  /  0.4553
43.466517996497345  /  0.4522833333333333  /  0.4553
43.46651711282228  /  0.4522833333333333  /  0.4553
43.46651640116619  /  0.4522833333333333  /  0.4553
43.46651594130347  /  0.4523  /  0.4553
43.46651518997735  /  0.4523  /  0.4553
43.46651503982857  /  0.4523  /  0.4553
end
```





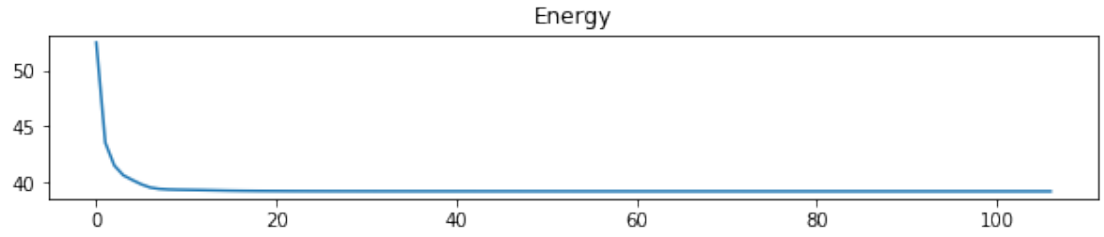


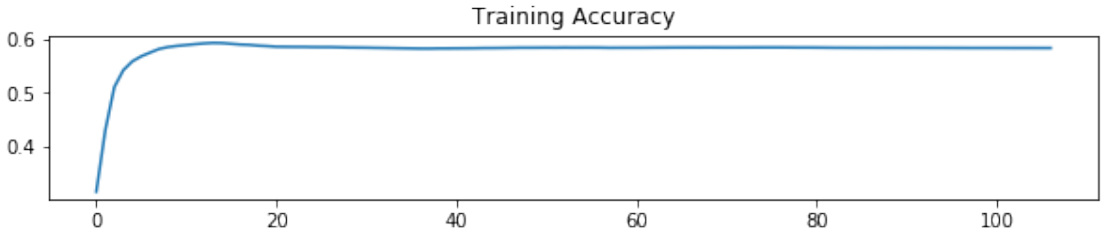

```
In [18]: initialize_centroid(10)
```

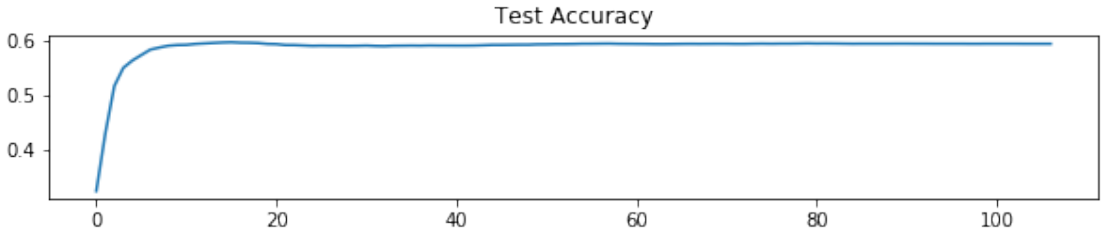
```
52.45211208748828  /  0.3173166666666667  /  0.3243
43.56948574325889  /  0.4309  /  0.4293
41.54328416298957  /  0.5105666666666666  /  0.5161
40.67733440624567  /  0.5418833333333334  /  0.5494
40.25718316525845  /  0.5582333333333334  /  0.5628
39.87223450064548  /  0.5676333333333333  /  0.5729
39.58539066164093  /  0.5745833333333333  /  0.5828
39.462151991579915  /  0.5812666666666667  /  0.5866
39.41564449718877  /  0.5847333333333333  /  0.5899
39.39448435260158  /  0.5868666666666666  /  0.5912
39.380795951120405  /  0.5884  /  0.5915
39.36790375615238  /  0.5900666666666666  /  0.5932
39.35321794947827  /  0.5914333333333334  /  0.5941
39.33723093697647  /  0.5923166666666667  /  0.595
39.319463472233764  /  0.5919666666666666  /  0.5958
39.30482057499896  /  0.59065  /  0.5963
39.294025270925296  /  0.5893  /  0.5955
39.28616376746184  /  0.58865  /  0.5953
```

39.27996212277627 / 0.58725 / 0.5947
39.275204159149126 / 0.5864166666666667 / 0.5932
39.27116725386133 / 0.58525 / 0.5928
39.26774298058673 / 0.5851333333333333 / 0.5912
39.26497384862857 / 0.5849666666666666 / 0.5911
39.26225725584082 / 0.5849333333333333 / 0.5904
39.25992271192712 / 0.58465 / 0.5897
39.258609952239496 / 0.5845666666666667 / 0.59
39.25771952434489 / 0.5845333333333333 / 0.5898
39.257031007009644 / 0.5841666666666666 / 0.5898
39.25634037555914 / 0.5837333333333333 / 0.5896
39.255717866272136 / 0.5836166666666667 / 0.5898
39.25511165798978 / 0.5834666666666667 / 0.5902
39.25459256189465 / 0.58315 / 0.5895
39.25415826454262 / 0.58295 / 0.5892
39.253817355758834 / 0.5826333333333333 / 0.5899
39.25353969810863 / 0.5823666666666667 / 0.59
39.25330743556131 / 0.5821166666666666 / 0.5902
39.25308798742076 / 0.58185 / 0.59
39.25288676788685 / 0.5818166666666666 / 0.5904
39.25270375730362 / 0.5821166666666666 / 0.5902
39.25250854635538 / 0.5821166666666666 / 0.5902
39.25232045966132 / 0.58225 / 0.5901
39.25210032114129 / 0.5823666666666667 / 0.5901
39.2517957508493 / 0.5824666666666667 / 0.5903
39.251490677992926 / 0.5827833333333333 / 0.5908
39.25127549868341 / 0.5828166666666666 / 0.5914
39.25107450686733 / 0.583 / 0.5914
39.25083152405032 / 0.5831666666666667 / 0.5916
39.25065043753163 / 0.5834333333333334 / 0.5918
39.25036548763163 / 0.5834166666666667 / 0.5918
39.24996050103901 / 0.5834833333333334 / 0.5924
39.24958316970627 / 0.5835666666666667 / 0.5925
39.249275867184814 / 0.58345 / 0.5929
39.24899838214348 / 0.58365 / 0.5929
39.24875980701871 / 0.5835166666666667 / 0.5931
39.24855547125616 / 0.5835666666666667 / 0.5935
39.24840853988725 / 0.5836166666666667 / 0.5935
39.24832647213891 / 0.5835166666666667 / 0.5937
39.248251230779466 / 0.58325 / 0.5938
39.248176986007316 / 0.5833 / 0.5934
39.248094888368314 / 0.58335 / 0.5933
39.24801861225011 / 0.5833166666666667 / 0.5932
39.24792859378626 / 0.5833833333333334 / 0.5931
39.247830697438815 / 0.5835 / 0.5929
39.24772678540762 / 0.5835666666666667 / 0.5928
39.24765405182047 / 0.5837166666666667 / 0.593
39.24761859282382 / 0.5837666666666667 / 0.5932

```
39.24759442238754  /  0.5838166666666667  /  0.5933
39.24757968266533  /  0.5837833333333333  /  0.5932
39.24756443952369  /  0.5838833333333333  /  0.5933
39.24753713893863  /  0.5838333333333333  /  0.5934
39.247513944841415  /  0.5837666666666667  /  0.5934
39.24749446285263  /  0.5837833333333333  /  0.5932
39.24746642566339  /  0.58385  /  0.5932
39.2474269796423  /  0.5838833333333333  /  0.5935
39.24739991500569  /  0.5839333333333333  /  0.5937
39.24737233843373  /  0.5839333333333333  /  0.5935
39.247348394744364  /  0.5839333333333333  /  0.5937
39.247312445823155  /  0.58385  /  0.5937
39.24727339664349  /  0.5837166666666667  /  0.5939
39.24723387947734  /  0.58375  /  0.5942
39.24718504562241  /  0.5836333333333333  /  0.5939
39.24712456295524  /  0.5835  /  0.594
39.24707848423756  /  0.58335  /  0.5939
39.24705015372208  /  0.5832833333333334  /  0.5937
39.2470371109618  /  0.5832333333333334  /  0.5935
39.247032007706  /  0.5831833333333334  /  0.5936
39.24702878256432  /  0.5831666666666667  /  0.5936
39.247026888798224  /  0.5831333333333333  /  0.5936
39.24702141594114  /  0.5831833333333334  /  0.5936
39.24701523472493  /  0.5832  /  0.5937
39.24700698736291  /  0.5831833333333334  /  0.5937
39.24700041050152  /  0.5831833333333334  /  0.5936
39.24699234839425  /  0.5831166666666666  /  0.5936
39.24698910832935  /  0.5830833333333333  /  0.5935
39.24698516837128  /  0.5830833333333333  /  0.5935
39.24698177646043  /  0.5830333333333333  /  0.5935
39.24697860016115  /  0.583  /  0.5935
39.2469757607686  /  0.58295  /  0.5934
39.24697234540435  /  0.58295  /  0.5934
39.24697034973042  /  0.5829333333333333  /  0.5935
39.24696931081209  /  0.58295  /  0.5935
39.24696866638922  /  0.5829333333333333  /  0.5935
39.24696676950887  /  0.5828833333333333  /  0.5935
39.246964478031465  /  0.5828666666666666  /  0.5934
39.24696305705296  /  0.58285  /  0.5934
39.246962356653626  /  0.5828333333333333  /  0.5934
39.24696168336116  /  0.5828333333333333  /  0.5934
end
```



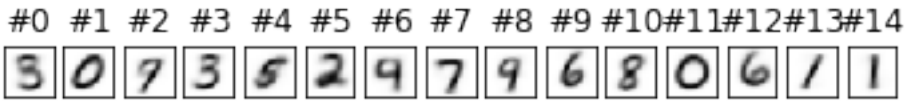


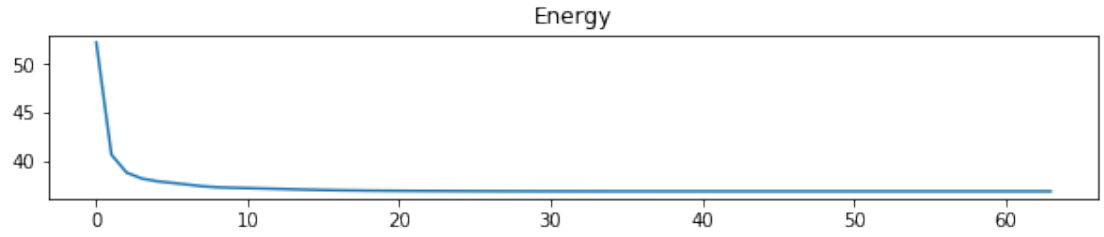


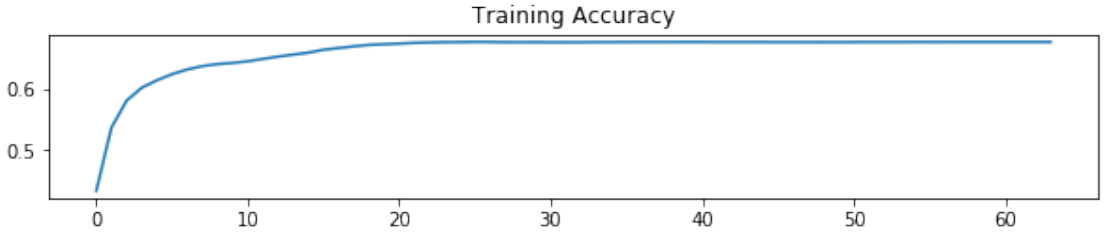

```
In [19]: initialize_centroid(15)

52.25625357743773  /  0.4339  /  0.4403
40.6094147942605  /  0.5368833333333334  /  0.5352
38.74551928616249  /  0.581  /  0.5794
38.12682594749652  /  0.6021166666666666  /  0.5995
37.844073414092634  /  0.6142166666666666  /  0.612
37.67592089258143  /  0.6242333333333333  /  0.6208
37.50153772198304  /  0.6318666666666667  /  0.6295
37.321231328382176  /  0.6373833333333333  /  0.6356
37.21613285644158  /  0.6406833333333334  /  0.6404
```

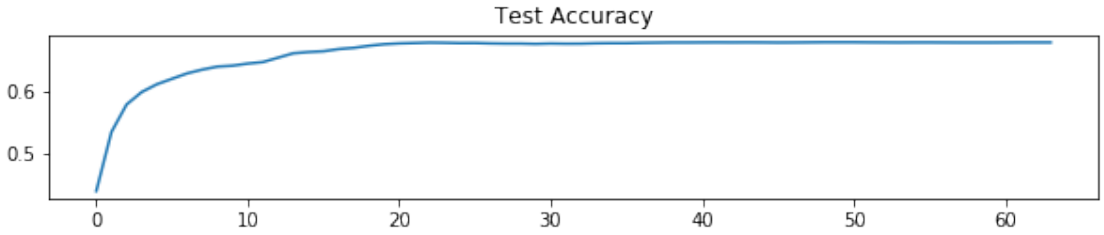
37.16495383833321 / 0.64255 / 0.6419
37.124615704109736 / 0.6452333333333333 / 0.6453
37.086966681461355 / 0.6491666666666667 / 0.6477
37.047805318601036 / 0.6529833333333334 / 0.6547
37.00603293692064 / 0.6561833333333333 / 0.6618
36.96627441452064 / 0.65945 / 0.6638
36.93148213224988 / 0.6641333333333334 / 0.6651
36.906082216271706 / 0.6669 / 0.6686
36.88523566280283 / 0.66965 / 0.6706
36.86848088866416 / 0.67215 / 0.6739
36.855952005401804 / 0.6731333333333334 / 0.6764
36.845923938697005 / 0.6742666666666667 / 0.6777
36.837676977846314 / 0.6754333333333333 / 0.6783
36.83053031378193 / 0.6760333333333334 / 0.679
36.82421483137984 / 0.6765166666666667 / 0.6787
36.81817123028979 / 0.6766333333333333 / 0.6782
36.812868155263665 / 0.6769833333333334 / 0.6783
36.808066320879306 / 0.6768333333333333 / 0.6777
36.80422613986008 / 0.6764833333333333 / 0.6774
36.80145428427102 / 0.6763833333333333 / 0.6774
36.79916945851882 / 0.6765 / 0.6769
36.79773377932824 / 0.6762833333333333 / 0.6775
36.79677538821487 / 0.6763333333333333 / 0.6772
36.79583142463407 / 0.67635 / 0.6773
36.79504901367844 / 0.6765666666666666 / 0.6779
36.79446860808208 / 0.6766166666666666 / 0.6781
36.79411106833437 / 0.6767166666666666 / 0.6781
36.793880882675005 / 0.6767833333333333 / 0.6786
36.79372025897091 / 0.6767666666666666 / 0.6789
36.793607228577685 / 0.6767666666666666 / 0.6792
36.793514635197326 / 0.6768833333333333 / 0.6792
36.79346322543467 / 0.67685 / 0.6793
36.79343784083789 / 0.67675 / 0.6794
36.793413043005486 / 0.6766666666666666 / 0.6793
36.79339840769376 / 0.6766833333333333 / 0.6794
36.79338328504882 / 0.6766333333333333 / 0.6793
36.793370325150974 / 0.6766166666666666 / 0.6791
36.79335689369569 / 0.6766 / 0.6792
36.79334830808652 / 0.67655 / 0.6794
36.79333783617226 / 0.6765166666666667 / 0.6796
36.79333335908865 / 0.67655 / 0.6796
36.79332504533373 / 0.6766333333333333 / 0.6796
36.79331272289316 / 0.6767 / 0.6794
36.79330605172628 / 0.6767 / 0.6793
36.79329754248843 / 0.67675 / 0.6792
36.7932914274612 / 0.6767833333333333 / 0.6793
36.79328523206633 / 0.6768166666666666 / 0.6793
36.79328154399675 / 0.6768166666666666 / 0.6792

```
36.79327990419406  /  0.6768333333333333  /  0.6791
36.79327940995133  /  0.6768833333333333  /  0.6791
36.7932780801792  /  0.6768833333333333  /  0.6791
36.7932770062857  /  0.6768833333333333  /  0.6792
36.79327599942751  /  0.6768666666666666  /  0.6793
36.793275651662206  /  0.6768833333333333  /  0.6793
36.793275069609194  /  0.6768833333333333  /  0.6793
end
```







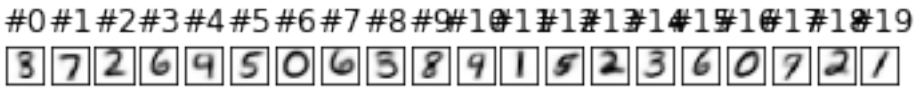


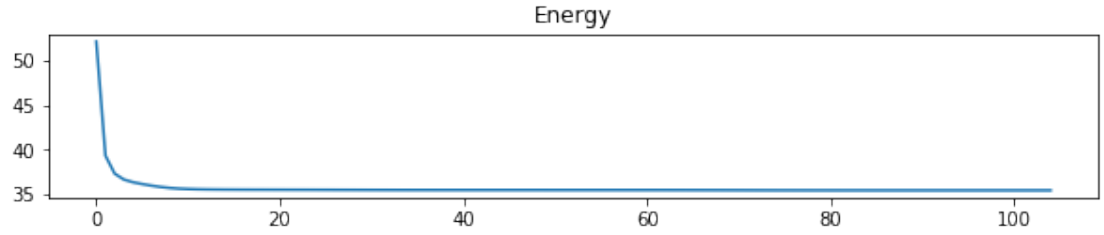

```
In [20]: initialize_centroid(20)
```

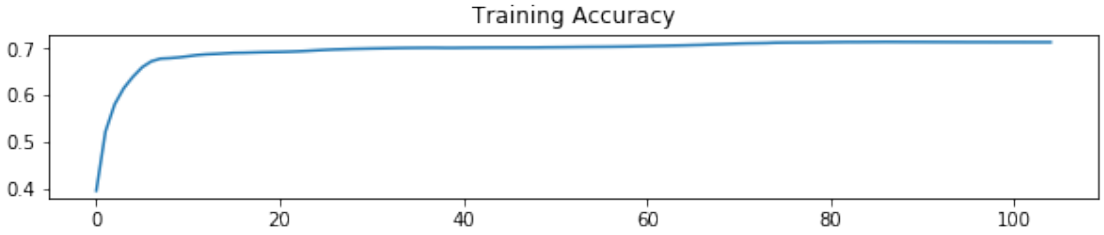
52.17459070502366 / 0.39605 / 0.4042
39.30485314325599 / 0.5224666666666666 / 0.5174
37.28114953413854 / 0.5807166666666667 / 0.5776
36.61384317299475 / 0.61535 / 0.6149
36.30837539611568 / 0.6397333333333334 / 0.6429
36.1032207095081 / 0.6600333333333334 / 0.6638
35.92454584702635 / 0.6726666666666666 / 0.6763
35.77313457628176 / 0.6784333333333333 / 0.6818
35.66119877184392 / 0.6794333333333333 / 0.6853
35.58567058539203 / 0.6810666666666667 / 0.6902
35.54160747936304 / 0.6835 / 0.6937
35.51407280979165 / 0.6860833333333334 / 0.6959
35.49806931863938 / 0.6876666666666666 / 0.6979
35.48797077308526 / 0.6884833333333333 / 0.699
35.48108520015344 / 0.6896166666666667 / 0.7002
35.47579522741864 / 0.6906666666666667 / 0.7005
35.47228349695429 / 0.6909166666666666 / 0.7018
35.469793667212244 / 0.6914833333333333 / 0.7011
35.4675999728486 / 0.6920333333333333 / 0.7018
35.465618993486714 / 0.69235 / 0.7023
35.46345709289674 / 0.69275 / 0.7021
35.461507432879245 / 0.6932333333333334 / 0.7024
35.459523724705804 / 0.69395 / 0.7028
35.457304738168666 / 0.6949333333333333 / 0.7033
35.454577278541144 / 0.6961 / 0.7039
35.451230547234545 / 0.6971666666666667 / 0.7051
35.44805196318204 / 0.6980333333333333 / 0.7061
35.445500907998884 / 0.6985666666666667 / 0.7069
35.44322434306319 / 0.6991833333333334 / 0.7074
35.4407644284375 / 0.69955 / 0.7081
35.43830800925088 / 0.7000166666666666 / 0.7081
35.43576976985663 / 0.7003333333333334 / 0.7084
35.43347984148503 / 0.7007166666666667 / 0.7087
35.43182656534693 / 0.7011 / 0.7082
35.43082414537321 / 0.7013333333333334 / 0.7079
35.43007887082432 / 0.7014833333333333 / 0.7074
35.42940974834184 / 0.70165 / 0.7077
35.42892747704419 / 0.7016333333333333 / 0.708
35.42865028343009 / 0.7014 / 0.7077
35.4284494572798 / 0.7014 / 0.7081
35.42824325061386 / 0.7016333333333333 / 0.7083
35.42805520727366 / 0.7017666666666666 / 0.7083
35.4279211246114 / 0.70185 / 0.708
35.427856841335895 / 0.7019666666666666 / 0.7083
35.42781246659882 / 0.702 / 0.7083
35.42775358614625 / 0.7020666666666666 / 0.7085
35.42770412004868 / 0.7021666666666667 / 0.7087
35.42764884831199 / 0.7021666666666667 / 0.709

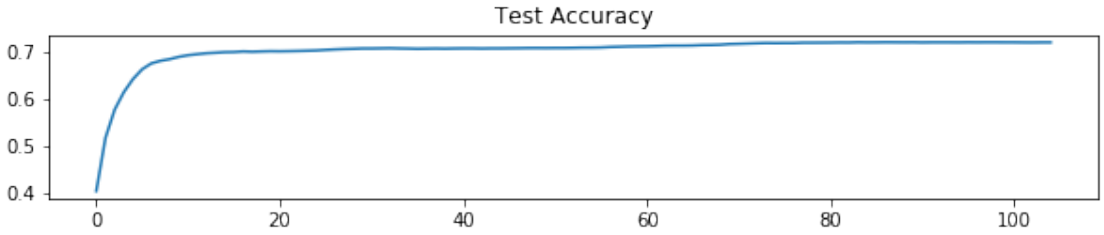
35.427587535338766 / 0.7023 / 0.7091
35.42750423739565 / 0.7025666666666667 / 0.709
35.42742254760963 / 0.7027166666666667 / 0.7092
35.427320748573365 / 0.7028666666666666 / 0.7092
35.4271942815384 / 0.7031 / 0.7095
35.427038319212926 / 0.70335 / 0.7099
35.426834669356595 / 0.7035166666666667 / 0.7099
35.42655584134004 / 0.7036333333333333 / 0.7103
35.426229914065324 / 0.7038 / 0.7113
35.4258251305108 / 0.7040666666666666 / 0.7118
35.42542167362465 / 0.7042833333333334 / 0.7124
35.425032593950945 / 0.7048 / 0.7127
35.42456202756207 / 0.7051166666666666 / 0.7129
35.42389630640004 / 0.7055833333333333 / 0.7134
35.42292582622857 / 0.7059 / 0.714
35.42165001201651 / 0.7063833333333334 / 0.7142
35.420255063842525 / 0.7067833333333333 / 0.7142
35.41848827362462 / 0.7074 / 0.7145
35.416373354298955 / 0.7078666666666666 / 0.7153
35.4140627355997 / 0.7087333333333333 / 0.7155
35.4118576417856 / 0.7092666666666667 / 0.7161
35.409751613099615 / 0.70995 / 0.7175
35.40741286899414 / 0.7106666666666667 / 0.718
35.4045493756123 / 0.71115 / 0.7186
35.40186364818562 / 0.7113833333333334 / 0.7192
35.399420504403295 / 0.7117666666666667 / 0.7197
35.39755855108116 / 0.7124666666666667 / 0.7195
35.39640330607988 / 0.7127 / 0.7197
35.39545041329833 / 0.7128333333333333 / 0.7197
35.39465483195485 / 0.7129333333333333 / 0.7204
35.39403788735119 / 0.7130166666666666 / 0.7204
35.393547604990026 / 0.7132 / 0.7205
35.3930458909596 / 0.71355 / 0.7206
35.392650061147336 / 0.7137166666666667 / 0.7209
35.39233874387978 / 0.7138666666666666 / 0.7208
35.39200510975389 / 0.7138333333333333 / 0.7213
35.39176789681299 / 0.7139666666666666 / 0.7211
35.39160579237462 / 0.71405 / 0.7212
35.39151099796675 / 0.7141833333333333 / 0.7214
35.39144602520788 / 0.7141166666666666 / 0.7213
35.39140469900587 / 0.714 / 0.7212
35.39137667186937 / 0.7139833333333333 / 0.7213
35.39135484416521 / 0.71395 / 0.721
35.3913389046365 / 0.7138666666666666 / 0.7212
35.39132625890493 / 0.7138166666666667 / 0.7212
35.39131335644935 / 0.7137666666666667 / 0.7212
35.39129228917831 / 0.7137166666666667 / 0.7211
35.39127468718252 / 0.7136166666666667 / 0.7213

```
35.3912650920471  /  0.7136333333333333  /  0.7212
35.39125889820547  /  0.7136  /  0.7213
35.39124934237087  /  0.7135666666666667  /  0.7213
35.39124229123311  /  0.7135833333333333  /  0.7212
35.39123744694309  /  0.7135833333333333  /  0.7212
35.39123359665282  /  0.7135666666666667  /  0.721
35.391231297664554  /  0.71355  /  0.7209
35.39122891960526  /  0.71355  /  0.7211
35.39122853700762  /  0.71355  /  0.7211
end
```









In [ ]: