(a)

We choose k = 37, since the letters C, I, J, K, L, M, O, P, S, U, V, W, X, Y, Z have similar uppercase and lowercase, their images will be quite similar and hence should be put in the same cluster.

(b)

First, we preprocess the datasets using prep.py:

```
#!/usr/bin/env python

import struct
import pickle

# SIZE: 124800 for training; 20800 for testing
# DIM: 28*28=784 pixels

train_images = "data/emnist-letters-train-images-idx3-ubyte"
train_labels = "data/emnist-letters-train-labels-idx1-ubyte"
test_images = "data/emnist-letters-test-images-idx3-ubyte"
test_labels = "data/emnist-letters-test-labels-idx1-ubyte"

def load_img(filepath):
    data = []
    print 'Loading: %s' % (filepath)
    with open(filepath, 'rb') as f:
        magic, size, rows, cols = struct.unpack('>IIII', f.read(16))
        for img in range(size):
            data.append(struct.unpack('>784B', f.read(784)))
            if (img+1) % 10000 == 0:
                print 'Loaded: %s' % (str(img+1))
    return data

def load_lbl(filepath):
    print 'Loading: %s' % (filepath)
    data = []
    with open(filepath, 'rb') as f:
        magic, size = struct.unpack('>II', f.read(8))
        for img in range(size):
            data.append(struct.unpack('>B', f.read(1))[0])
            if (img+1) % 10000 == 0:
                print 'Loaded: %s' % (str(img+1))
    return data

def write_to_file(filepath, data):
    print 'Writing %s to file...' % (filepath)
    with open(filepath, 'w') as f:
        for i, img in enumerate(data):
            f.write(str(i))
            if (i+1) % 10000 == 0:
                print 'Writing img: %s' % (str(i+1))
            for pixel in img:
                f.write(' ' + str(pixel))
            f.write('\n')

def write_to_pkl(filepath, data):
    print 'Writing %s to pkl...' % (filepath)
    with open(filepath, 'wb') as f:
        pickle.dump(data, f)

write_to_file("train_images", load_img(train_images))
write_to_pkl("train_labels.pkl", load_lbl(train_labels))
write_to_file("test_images", load_img(test_images))
```

```python
write_to_pkl("test_labels.pkl", load_lbl(test_labels))
```

We use init_cens.py to generate k = 37 random centroids:

```python
#!/usr/bin/env python

import numpy as np
import pickle

def init_centroids(k = 37, dim = 28*28):
    cens = {}
    for i in range(k):
        cens[i] = np.random.uniform(low=0, high=256, size=(dim,))
    return cens

init_cens = init_centroids()
with open('init_cens.pkl', 'wb') as f:
    pickle.dump(init_cens, f)
```

For each iteration of k-means algorithm, we run the following MapReduce job:

mapper.py

```python
#!/usr/bin/env python
import sys
import numpy as np
import math
import pickle

def euclid_dist(v1, v2):
    return math.sqrt(np.dot(v1-v2, v1-v2))

# Get the list of old centroids
with open('old_cens.pkl', 'rb') as f:
    cens = pickle.load(f)

partial_sum = {}

for line in sys.stdin:
    dataline = line.strip().split()
    # First element is only for numbering the images
    img = np.asarray(dataline[1:]).astype(int)

    # Get the closest centroid
    min_dist = 1e10 # Magic number for inf
    for i, cen in cens.items():
        dist = euclid_dist(img, cen)
        if dist < min_dist:
            min_dist = dist
            assigned_cen = i

    # Add the current sample to the partial sum
    if assigned_cen not in partial_sum:
        partial_sum[assigned_cen] = {'count': 1, 'img': img}
    else:
        partial_sum[assigned_cen]['count'] += 1
        partial_sum[assigned_cen]['img'] += img

# Print the partial sums from this mapper
for cen in partial_sum:
    img = ' '.join([str(pixel) for pixel in partial_sum[cen]['img']])
    print '%s\t%s %s' % (str(cen), str(partial_sum[cen]['count']), img)
```

reducer.py

```
#!/usr/bin/env python

import sys
import numpy as np

curr_cen = None

for line in sys.stdin:
    dataline = line.strip().split()
    img = np.asarray(dataline).astype(int)
    cen = img[0]
    count = img[1]

    if curr_cen == None:
        curr_count = count
        curr_sum = img[2:]
        curr_cen = cen
    elif curr_cen == cen:
        # Summing points in the same cluster
        curr_sum += img[2:]
        curr_count += count
    else:
        # Print out the new centroids: cen + pixel
        avg = ' '.join([str(round(float(pixel)/curr_count, 3)) for pixel in curr_sum])
        print '%s\t%s' % (str(curr_cen), avg)

        curr_count = count
        curr_sum = img[2:]
        curr_cen = cen

# Print out the new centroids
avg = ' '.join([str(round(pixel/curr_count, 3)) for pixel in curr_sum])
print '%s\t%s' % (str(curr_cen), avg)
```

At the end of each round, we check whether the total distances between the old and the new centroids are below a certain threshold using check.py:

```
#!/usr/bin/env python

import numpy as np
import math
import pickle

with open('old_cens.pkl', 'rb') as f:
    old_cens = pickle.load(f)

# Convert new_cens to numpy array, and dump it to pkl file
new_cens = {}
with open("new_cens", 'r') as f:
    for line in f.readlines():
        dataline = line.strip().split()
        cen = int(dataline[0])
        new_cens[cen] = np.asarray(dataline[1:]).astype(float)
# If centroids not updated
for cen in old_cens:
    if cen not in new_cens:
        new_cens[cen] = old_cens[cen]

# Write new centroids to pickle file
with open('new_cens.pkl', 'wb') as f:
    pickle.dump(new_cens, f)
```

```python

# Calculate total distance between corresponding old and new centroids
diff = 0
for cen in new_cens:
    v1 = new_cens[cen]
    v2 = old_cens[cen]
    diff += math.sqrt(np.dot(v1-v2, v1-v2))

# Logging the diff
with open("diff.log", 'a+') as f:
    f.write(str(diff) + '\n')

# Set stopping threshold
THRESHOLD = 30

# If diff falls below given threshold, stop
if diff <= THRESHOLD:
    print 'STOP'
```

We wrap the shell commands into mr.sh, and limit the number of iterations to at most 150 (to avoid running for too long):

```bash
#!/bin/bash
rm new_cens.pkl old_cens.pkl init_cens.pkl diff.log new_cens
python init_cens.py
cp init_cens.pkl old_cens.pkl

MAX_ITER=150

for ITER in {1..150}
do
    hdfs dfs -rm -r out
    hadoop jar /usr/hdp/2.4.2.0-258/hadoop-mapreduce/hadoop-streaming.jar -D mapred.job.name="Round_$ITER/$MAX_ITER" -D mapred.map.tasks=21 -D mapred.reduce.tasks=7 -file mapper.py -mapper mapper.py -file reducer.py -reducer reducer.py -input train_images -output out -file old_cens.pkl
    rm new_cens
    hdfs dfs -cat out/* >> new_cens
    if [[ $(python check.py) == "STOP" ]]
    then
        printf "CONVERGED: BELOW THRESHOLD\n"
        break
    else
        rm old_cens.pkl
        mv new_cens.pkl old_cens.pkl
    fi
done
mv new_cens final_cens
rm new_cens.pkl old_cens.pkl
hdfs dfs -rm -r out
hadoop jar /usr/hdp/2.4.2.0-258/hadoop-mapreduce/hadoop-streaming.jar -D mapred.job.name="Assign" -D mapred.map.tasks=10 -D mapred.reduce.tasks=2 -file assign_mapper.py -mapper assign_mapper.py -file assign_reducer.py -reducer assign_reducer.py -input train_images -output out -file final_cens
hdfs dfs -cat out/* >> tmp
sort -k1 -n tmp >> assign
rm tmp
python accuracy.py
```

Finally, we use the centroids from the algorithm to assign the data points accordingly and output the statistics using a simple MapReduce job:

assign_mapper.py

```
#!/usr/bin/env python

import numpy as np
import math
import sys

def euclid_dist(v1, v2):
    return math.sqrt(np.dot(v1-v2, v1-v2))

# Get the list of final centroids
cens = {}
with open('final_cens', 'r') as f:
    for line in f.readlines():
        dataline = line.strip().split()
        cens[int(dataline[0])] = np.asarray(dataline[1:]).astype(float)

for line in sys.stdin:
    dataline = line.strip().split()
    number = dataline[0]
    dataline = dataline[1:]
    img = np.asarray(dataline).astype(int)

    # Get the closest centroid
    min_dist = 1e10 # Magic number for inf
    for i, cen in cens.items():
        dist = euclid_dist(cen, img)
        if dist < min_dist:
            min_dist = dist
            assigned_cen = i

    print '%s\t%s' % (number, str(assigned_cen))
```

assign_reducer.py

```
#!/usr/bin/env python

import sys

for line in sys.stdin:
    dataline = line.strip().split()
    for c in dataline:
        print '%s' % (c),
    print ''
```

(c)

We use accuracy.py to calculate accuracy for each output of the corresponding random seeds:

```
#!/usr/bin/env python

import pickle

# Load labels
with open("train_labels.pkl", 'rb') as f:
    labels = pickle.load(f)

def file_to_dict(path):
    dict = {}
    with open(path, 'r') as f:
        for line in f.readlines():
            dataline = line.strip().split()
            dict[int(dataline[0])] = int(dataline[1])
    return dict
```

```

NUM_CLUSTER = 37
NUM_LABEL = 26

# Load cluster assignment
path = 'assign'

assignment = file_to_dict(path)
# Record clusters and the assigned img with their true label
# Note: EMNIST labels start from 1, not 0
count_cluster = {cluster: {label: [] for label in range(1, NUM_LABEL+1)}\
                 for cluster in range(NUM_CLUSTER)}

# Loop through all assigned images
for i, c in assignment.items():
    print 'Count: %s' % (str(i))
    # Increase the corr. label count of current cluster by 1
    # Note: EMNIST labels start from 1, not 0
    count_cluster[c][labels[i]].append(i)

# Get the cluster label and calculate its accuracy
accuracy = {cluster: {'label': -1, 'total': 0, 'correct': 0, 'acc': -1}\
                    for cluster in range(NUM_CLUSTER)}
for c, cluster in count_cluster.items():
    print 'Calc acc: %s' % (str(c))
    max_label = -1
    best_label = -1
    for l, label in cluster.items():
        accuracy[c]['total'] += len(label)
        if len(label) > max_label:
            max_label = len(label)
            best_label = l
    # Note: EMNIST labels start from 1, not 0
    accuracy[c]['label'] = best_label
    accuracy[c]['correct'] = max_label
    if accuracy[c]['total'] == accuracy[c]['correct'] and accuracy[c]['total'] == 0:
        accuracy[c]['acc'] = -1
    else:
        accuracy[c]['acc'] = round(float(accuracy[c]['correct']) / accuracy[c]['total'] * 100, 2)

with open('accuracy', 'w') as f:
    total = 0
    correct = 0
    for c, cluster in accuracy.items():
        total += cluster['total']
        correct += cluster['correct']
        f.write(str(c) + ' ' + str(cluster['total']) + ' ' + str(cluster['label']) +
                ' ' + str(cluster['correct']) + ' ' + str(cluster['acc']) + '\n')
    f.write("TOTAL: " + str(total) + '\n' +  "CORRECT: " + str(correct) + '\n' + "OVERALL ACCURACY: " + str(round(float(correct)/total*100, 2)) + '\n')
```

We report the corresponding statistics for each random seed in the following tables:

Table. 1. The Accuracy of Clustering Performance with Random Seed 1

Cluster Number	# train images belongs to the cluster	Label of the cluster	# correctly clustered images	Classification Accuracy (%)
0	0	NA	0	NA
1	4938	10	1743	35.3
2	5488	24	2298	41.87
3	3671	19	2878	78.4
4	0	NA	0	NA
5	4350	18	2324	53.43
6	6103	6	1647	26.99
7	4215	8	1297	30.77
8	0	NA	0	NA
9	0	NA	0	NA
10	3152	26	2372	75.25
11	3799	13	3353	88.26
12	3059	14	1701	55.61
13	4181	16	1545	36.95
14	0	NA	0	NA
15	3562	15	1695	47.59
16	4204	4	1352	32.16
17	3955	21	2023	51.15
18	4946	3	1196	24.18
19	5073	23	1022	20.15
20	3485	11	2001	57.42
21	4838	26	1140	23.56
22	4083	15	2089	51.16
23	4208	5	2144	50.95
24	5239	10	1252	23.9
25	0	NA	0	NA
26	0	NA	0	NA
27	3701	17	1321	35.69
28	0	NA	0	NA
29	4230	16	811	19.17
30	6692	9	2194	32.79
31	4212	8	1325	31.46
32	3688	23	2869	77.79
33	3718	22	2267	60.97
34	0	NA	0	NA
35	4789	14	1264	26.39
36	7221	9	1747	24.19
Total Set	124800	NA	50870	40.76

Table. 2. The Accuracy of Clustering Performance with Random Seed 2

Cluster Number	# train images belongs to the cluster	Label of the cluster	# correctly clustered images	Classification Accuracy (%)
0	0	NA	0	NA
1	5964	9	2110	35.38
2	3077	19	2409	78.29
3	3725	25	1042	27.97
4	4467	23	3527	78.96
5	3642	1	831	22.82
6	0	NA	0	NA
7	3736	4	1766	47.27
8	2804	22	1820	64.91
9	5054	6	1477	29.22
10	4227	5	1881	44.5
11	2468	21	847	34.32
12	4808	10	1871	38.91
13	3002	6	1073	35.74
14	2580	26	1500	58.14
15	2795	17	996	35.64
16	3989	18	2242	56.2
17	3848	3	1007	26.17
18	2826	14	1731	61.25
19	2380	26	1832	76.97
20	4181	15	2431	58.14
21	3304	11	1943	58.81
22	3890	16	1806	46.43
23	6536	9	1713	26.21
24	3320	19	1160	34.94
25	5193	24	2275	43.81
26	3308	8	915	27.66
27	3453	8	1087	31.48
28	4865	14	1223	25.14
29	3867	15	930	24.05
30	2401	22	1493	62.18
31	4682	13	3768	80.48
32	3301	21	1898	57.5
33	4094	16	789	19.27
34	3013	8	931	30.9
35	0	NA	0	NA
36	0	NA	0	NA
Total Set	124800	NA	54324	43.53

Table. 3. The Accuracy of Clustering Performance with Random Seed 3

Cluster Number	# train images belongs to the cluster	Label of the cluster	# correctly clustered images	Classification Accuracy (%)
0	0	NA	0	NA
1	3551	22	1326	37.34
2	3774	1	844	22.36
3	3577	21	1820	50.88
4	3680	3	954	25.92
5	4617	23	3551	76.91
6	4383	10	1448	33.04
7	3220	19	2467	76.61
8	6394	9	1763	27.57
9	2197	26	1692	77.01
10	4767	13	3776	79.21
11	2573	21	846	32.88
12	4028	8	1213	30.11
13	4914	14	1221	24.85
14	3422	11	2008	58.68
15	3708	16	1501	40.48
16	4053	8	1405	34.67
17	2178	26	1619	74.33
18	4160	18	2300	55.29
19	3460	25	1240	35.84
20	0	NA	0	NA
21	0	NA	0	NA
22	2557	22	1784	69.77
23	0	NA	0	NA
24	2895	10	1266	43.73
25	0	NA	0	NA
26	3325	15	1966	59.13
27	5235	24	2131	40.71
28	5868	16	1634	27.85
29	3043	14	1726	56.72
30	3200	4	1187	37.09
31	4128	16	870	21.08
32	6413	9	2107	32.86
33	3984	15	1779	44.65
34	3218	17	1187	36.89
35	3992	3	785	19.66
36	4286	5	2254	52.59
Total Set	124800	NA	53670	43

Best random seed: Random seed 2 and 3 have fewer clusters with no member compared to random seed 1, hence, together with better overall accuracy, they are somewhat better than random seed 1. Random seed 2 has slightly better overall accuracy (43.53%) compared to random seed 3 (43%), hence random seed 2 is the best seed (but only being better than seed 3 by a very small margin).

(d)

We will use the code of part (b) and (c), in particular the assign_mapper.py & assign_reducer.py (MapReduce job for assignment) and accuracy.py, to determine the accuracy of the model with random seed 2 on test set. Run the following shell script with corresponding input, which are the test dataset and the final centroids with random seed 2:

```
hdfs dfs -rm -r out
hadoop jar /usr/hdp/2.4.2.0-258/hadoop-mapreduce/hadoop-streaming.jar -D mapred.job.name="Assign" -D mapred.map.tasks=10 -D mapred.reduce.tasks=2 -file assign_mapper.py -mapper assign_mapper.py -file assign_reducer.py -reducer assign_reducer.py -input test_images -output out -file final_cens
hdfs dfs -cat out/* >> tmp
sort -k1 -n tmp >> assign
rm tmp
python accuracy.py
```

We report the accuracy in the following table:

Table. 4. The Accuracy of Clustering Performance on the test data

Cluster Number	# train images belongs to the cluster	Label of the cluster	# correctly clustered images	Classification Accuracy (%)
0	0	NA	0	NA
1	1040	9	350	33.65
2	523	19	410	78.39
3	613	25	176	28.71
4	768	23	613	79.82
5	607	1	154	25.37
6	0	NA	0	NA
7	605	4	302	49.92
8	497	22	311	62.58
9	846	6	242	28.61
10	703	5	326	46.37
11	423	21	134	31.68
12	774	10	306	39.53
13	490	6	165	33.67
14	426	26	250	58.69
15	462	17	180	38.96
16	617	18	348	56.4
17	608	3	157	25.82
18	479	14	289	60.33

19	414	26	325	78.5
20	702	15	420	59.83
21	539	11	312	57.88
22	641	16	313	48.83
23	1074	9	287	26.72
24	606	19	190	31.35
25	902	24	389	43.13
26	568	8	160	28.17
27	568	8	186	32.75
28	832	14	219	26.32
29	590	3	145	24.58
30	396	22	248	62.63
31	752	13	623	82.85
32	560	21	334	59.64
33	683	1	130	19.03
34	492	8	131	26.63
35	0	NA	0	NA
36	0	NA	0	NA
Total Set	20800	NA	9125	43.87