18CS10069_Assignment1_Q9

September 15, 2021

Name: Siba Smarak Panigrahi

Roll No.: 18CS10069

NOTE: The answers to the questions can be found in the text of the this document.

[1]:
```
# import and loading data
import numpy as np
from matplotlib import pyplot
from keras.datasets import mnist
from collections import defaultdict, Counter
from sklearn.cluster import KMeans
from sklearn.metrics import accuracy_score
import warnings

warnings.filterwarnings("ignore")
np.random.seed(0)

(trainX, trainy), (testX, testy) = mnist.load_data()
print('Train: X=%s, y=%s' % (trainX.shape, trainy.shape))
print('Test: X=%s, y=%s' % (testX.shape, testy.shape))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
11501568/11490434 [==============================] - 0s 0us/step
Train: X=(60000, 28, 28), y=(60000,)
Test: X=(10000, 28, 28), y=(10000,)
```

[2]:
```
# create the training dataset
def create_dataset(train_labels, num=100):
    image_ids = defaultdict(lambda : 0)
    final_images, final_labels = [], []

    for i in range(len(train_labels)):
        label = train_labels[i]
        if image_ids[label] < num:
```

```python
            final_images.append(i)
            final_labels.append(label)
            image_ids[label] += 1

        if sum(list(image_ids.values())) == 10 * num:
            assert final_labels == [train_labels[x] for x in final_images]
            return final_images, final_labels

# vectorization of images
def vectorization(final_images, trainX):
    vector_length = trainX[0].shape[0] * trainX[0].shape[1]
    final_trainX = np.zeros((len(final_images), vector_length))
    for i in range(len(final_images)):
        id = final_images[i]
        final_trainX[i] = trainX[i].reshape(1, -1)/255.0
    return final_trainX

# create a deterministic test data and find accuacy on it
def test_accuracy(clf, cluster_label, testX, testy, num=50, verbose=False):
    final_testX = np.zeros((num, testX[0].shape[0] * testX[0].shape[1]))
    final_testy, ids = [], []

    for i in range(num):
        id = np.random.randint(len(testy))
        ids.append(id)
        final_testX[i] = testX[id].reshape(1, -1)/255.0
        final_testy.append(testy[id])

    preds = [cluster_label[x] for x in clf.predict(final_testX)]
    labels = np.array(final_testy)
    if verbose:
        print(labels)
        print(preds)
    acc = accuracy_score(labels, preds)
    return acc

# map cluster labels to actual range of class labels (for finding accuracy)
def map_label_to_class(kmeans_labels, actual_labels):
    default_map = defaultdict(lambda : [])
    for i in range(len(kmeans_labels)):
        kmean_label = kmeans_labels[i]
        actual_label = actual_labels[i]
        default_map[kmean_label].append(actual_label)

    mapper = {}

    for k,v in (default_map.items()):
```

```
        mapper[k] = Counter(v).most_common(1)[0][0]

    return mapper
```

[3]:
```
final_images, final_labels = create_dataset(trainy)
final_trainX = vectorization(final_images, trainX)
print(f'+++++ N for this problem: {final_trainX.shape[0]}')
print(f'+++++ n for this problem: {final_trainX.shape[1]}')
```

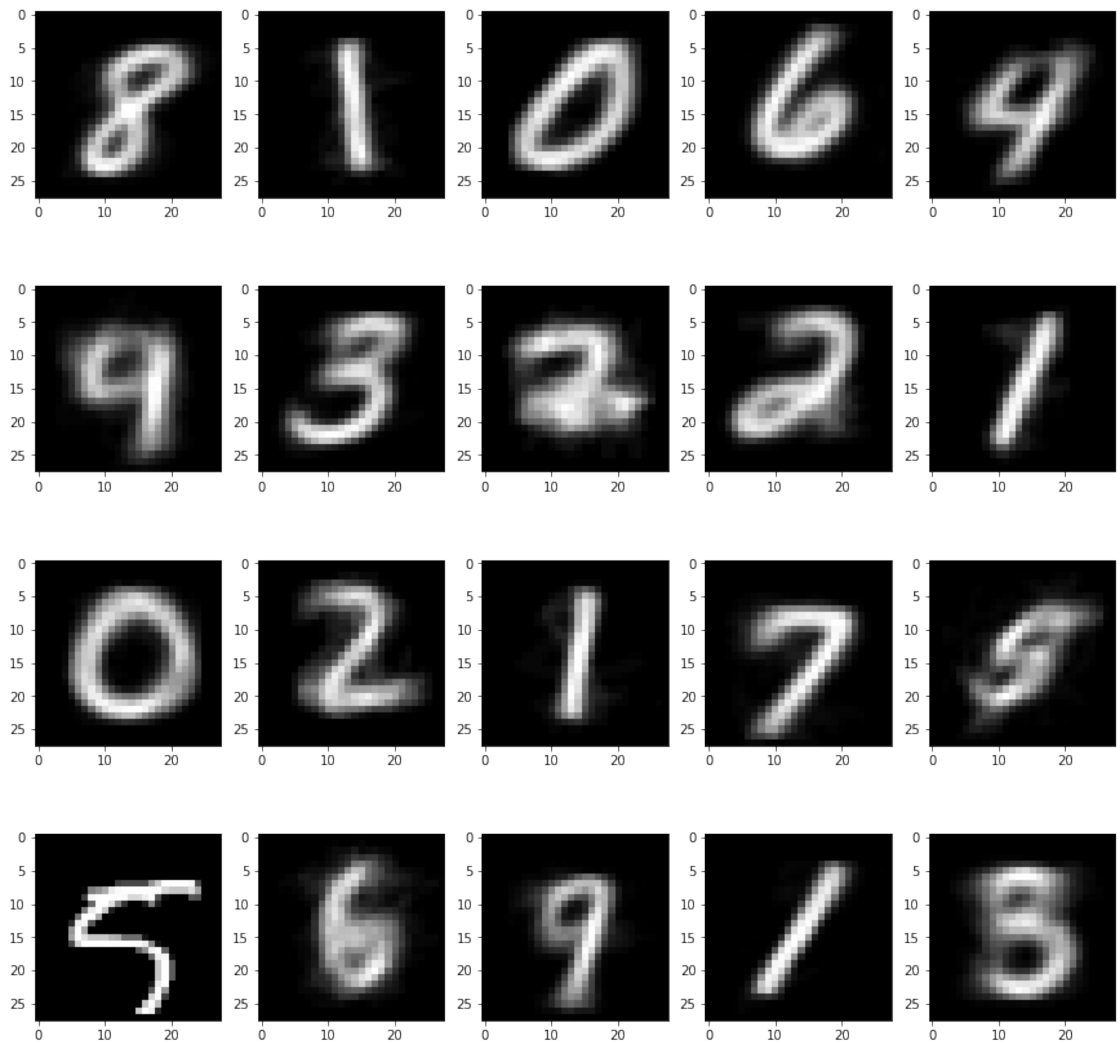
```
+++++ N for this problem: 1000
+++++ n for this problem: 784
```

0.0.1 Ans 9 (a) (i)

[4]:
```
# randomly initializing the centroids
convergence = 1e-4
n_clusters = 20
random_init = np.random.random_sample(size=(n_clusters, final_trainX.shape[1]))
kmeans_rand_init = KMeans(init=random_init, tol=convergence,␣
 ↪n_clusters=n_clusters, n_init=1, random_state=0)
kmeans_rand_init = kmeans_rand_init.fit(final_trainX)
print(f'+++++ Number of iterations required to converge: {kmeans_rand_init.
 ↪n_iter_}')
print(f'+++++ Jclust for {n_clusters} clusters: {kmeans_rand_init.inertia_/1000:
 ↪.4f}')
```

```
+++++ Number of iterations required to converge: 30
+++++ Jclust for 20 clusters: 32.4346
```

[5]:
```
# plotting the centroids
fig, ax = pyplot.subplots(4, 5)
fig.set_figheight(15)
fig.set_figwidth(15)
for i in range(n_clusters):
    x, y = i // 5, i % 5
    ax[x][y].imshow(kmeans_rand_init.cluster_centers_[i].reshape(28,28),␣
 ↪cmap=pyplot.get_cmap('gray'))
```



0.0.2 Ans 9 (b) (i)

[6]:
```
# finding test accuracy
mapper = map_label_to_class(kmeans_rand_init.labels_, trainy)
test_acc = test_accuracy(kmeans_rand_init, mapper, testX, testy, verbose=False)
print(f'+++++ Test Accuracy: {test_acc}')
```

```
+++++ Test Accuracy: 0.7
```

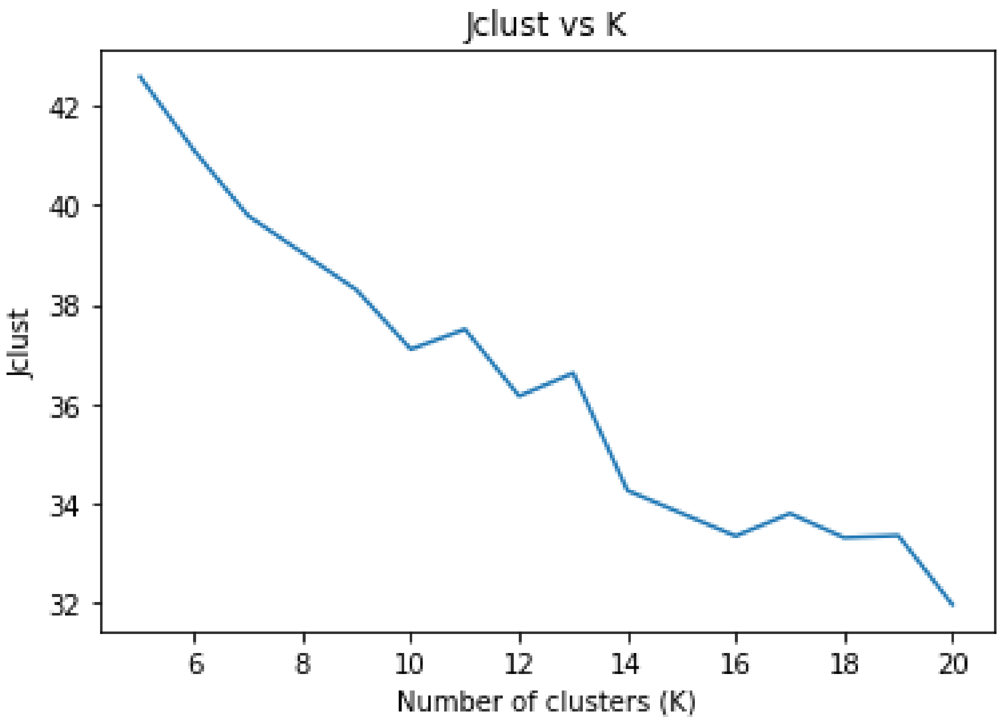
0.0.3 Ans 9 (c) (i)

[7]:
```
convergence = 1e-4
n_clusters = range(5,21)
Jclust_dict = {}
for n in n_clusters:
```

```
    random_init = np.random.random_sample(size=(n, final_trainX.shape[1]))
    kmeans_rand_init = KMeans(init=random_init, tol=convergence, n_clusters=n,␣
 ↪n_init=1, random_state=0)
    kmeans_rand_init = kmeans_rand_init.fit(final_trainX)
    print(f'+++++ Number of clusters: {n} ----- Jclust: {kmeans_rand_init.
 ↪inertia_/1000:.4f}')
    Jclust_dict[n] = kmeans_rand_init.inertia_/1000

pyplot.plot(list(Jclust_dict.keys()), list(Jclust_dict.values()));
pyplot.title(f'Jclust vs K');
pyplot.xlabel(f'Number of clusters (K)');
pyplot.ylabel(f'Jclust');
```

```
+++++ Number of clusters: 5 ----- Jclust: 42.5833
+++++ Number of clusters: 6 ----- Jclust: 41.0973
+++++ Number of clusters: 7 ----- Jclust: 39.7806
+++++ Number of clusters: 8 ----- Jclust: 39.0343
+++++ Number of clusters: 9 ----- Jclust: 38.2892
+++++ Number of clusters: 10 ----- Jclust: 37.0966
+++++ Number of clusters: 11 ----- Jclust: 37.5004
+++++ Number of clusters: 12 ----- Jclust: 36.1571
+++++ Number of clusters: 13 ----- Jclust: 36.6217
+++++ Number of clusters: 14 ----- Jclust: 34.2588
+++++ Number of clusters: 15 ----- Jclust: 33.7985
+++++ Number of clusters: 16 ----- Jclust: 33.3401
+++++ Number of clusters: 17 ----- Jclust: 33.7954
+++++ Number of clusters: 18 ----- Jclust: 33.3076
+++++ Number of clusters: 19 ----- Jclust: 33.3514
+++++ Number of clusters: 20 ----- Jclust: 31.9619
```



[10]:
```
# Optimal cluster (checking if the consecutive difference is less than some␣
↪epsilon, here say 0.001)
Jclust_values = list(Jclust_dict.values())
epsilon = 1e-3
found = False
for i in range(5, len(Jclust_values) + 4):
    if Jclust_values[i - 5] - Jclust_values[i - 4] <= epsilon:
        print(f'+++++ Best number of clusters could be considered as: {i}')
        found = True
        break

if not found:
    print(f'+++++ Best number of clusters could be considered as:␣
↪{len(Jclust_values) + 4}')
```

```
+++++ Best number of clusters could be considered as: 10
```

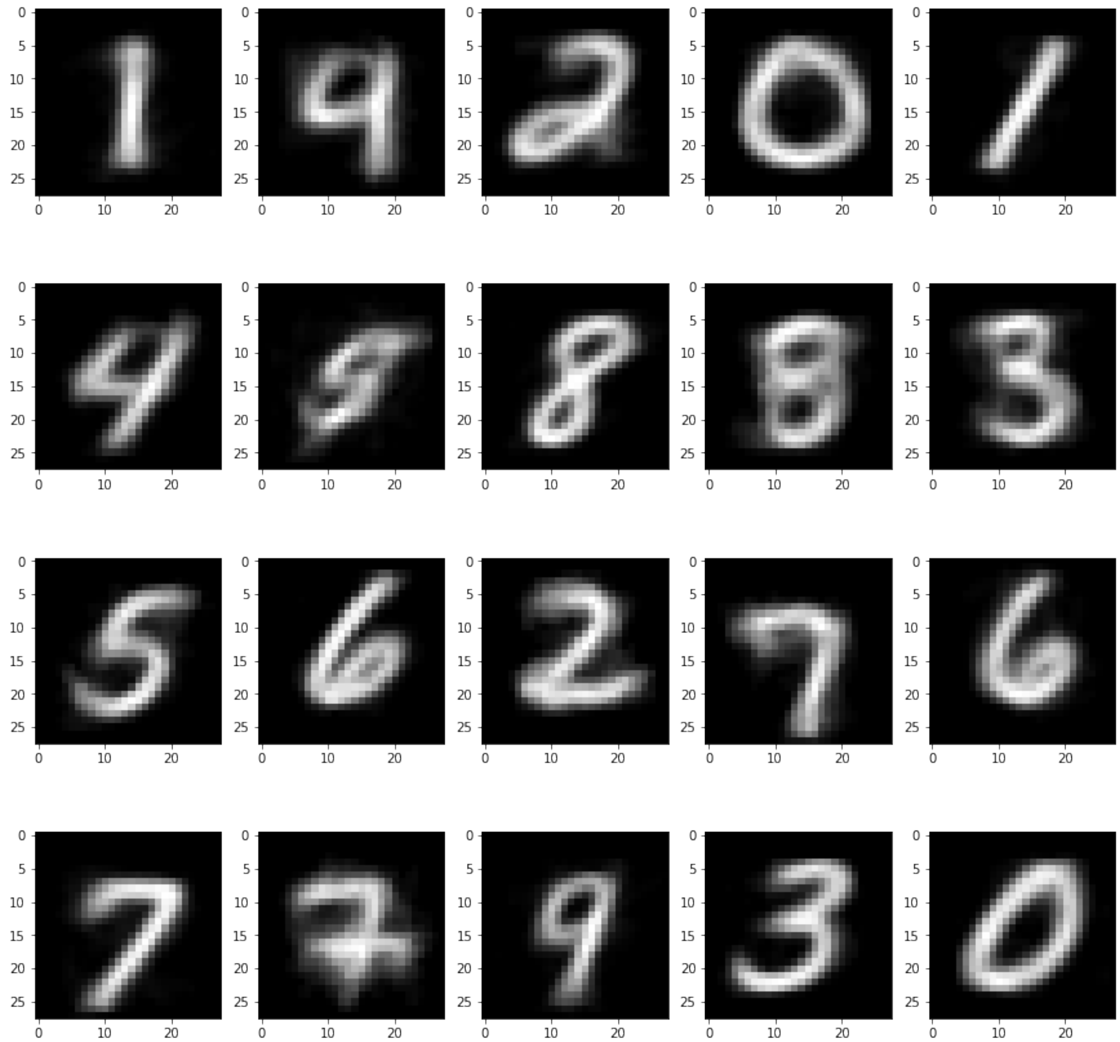
0.0.4 Ans 9 (a) (ii)

[11]:
```
convergence = 1e-4
n_clusters = 20
kmeans = KMeans(init='random', tol=convergence, n_clusters=n_clusters,␣
↪random_state=0)
kmeans = kmeans.fit(final_trainX)
```

```
print(f'+++++ Number of iterations required to converge: {kmeans.n_iter_}')
print(f'+++++ Jclust for {n_clusters} clusters: {kmeans.inertia_/1000:.4f}')
```

```
+++++ Number of iterations required to converge: 16
+++++ Jclust for 20 clusters: 31.8194
```

[12]:
```
fig, ax = pyplot.subplots(4, 5)
fig.set_figheight(15)
fig.set_figwidth(15)
for i in range(n_clusters):
    x, y = i // 5, i % 5
    ax[x][y].imshow(kmeans.cluster_centers_[i].reshape(28,28), cmap=pyplot.
 ↪get_cmap('gray'))
```



0.0.5 Ans 9 (b) (ii)

[13]:
```
mapper = map_label_to_class(kmeans.labels_, trainy)
test_acc = test_accuracy(kmeans, mapper, testX, testy, verbose=False)
print(f'+++++ Test Accuracy: {test_acc}')
```

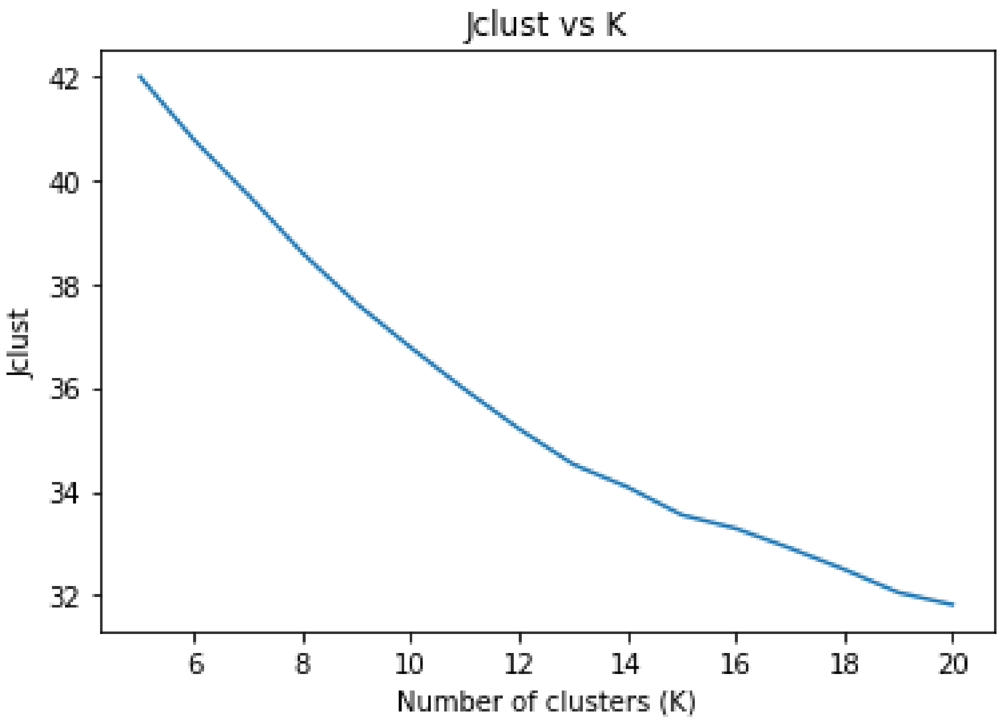
```
+++++ Test Accuracy: 0.72
```

0.0.6 Ans 9 (c) (ii)

[14]:
```
convergence = 1e-4
n_clusters = range(5,21)
Jclust_dict = {}
for n in n_clusters:
    kmeans = KMeans(init='random', tol=convergence, n_clusters=n,␣
 ↪random_state=0)
    kmeans = kmeans.fit(final_trainX)
    print(f'+++++ Number of clusters: {n} ----- Jclust: {kmeans.inertia_/1000:.
 ↪4f}')
    Jclust_dict[n] = kmeans.inertia_/1000

pyplot.plot(list(Jclust_dict.keys()), list(Jclust_dict.values()));
pyplot.title(f'Jclust vs K');
pyplot.xlabel(f'Number of clusters (K)');
pyplot.ylabel(f'Jclust');
```

```
+++++ Number of clusters: 5 ----- Jclust: 41.9870
+++++ Number of clusters: 6 ----- Jclust: 40.7663
+++++ Number of clusters: 7 ----- Jclust: 39.7054
+++++ Number of clusters: 8 ----- Jclust: 38.5886
+++++ Number of clusters: 9 ----- Jclust: 37.6163
+++++ Number of clusters: 10 ----- Jclust: 36.7673
+++++ Number of clusters: 11 ----- Jclust: 35.9599
+++++ Number of clusters: 12 ----- Jclust: 35.2025
+++++ Number of clusters: 13 ----- Jclust: 34.5191
+++++ Number of clusters: 14 ----- Jclust: 34.0852
+++++ Number of clusters: 15 ----- Jclust: 33.5479
+++++ Number of clusters: 16 ----- Jclust: 33.2865
+++++ Number of clusters: 17 ----- Jclust: 32.9057
+++++ Number of clusters: 18 ----- Jclust: 32.4967
+++++ Number of clusters: 19 ----- Jclust: 32.0533
+++++ Number of clusters: 20 ----- Jclust: 31.8194
```



[15]:
```
# Optimal cluster (checking if the consecutive difference is less than some
↪epsilon, here say 0.001)
Jclust_values = list(Jclust_dict.values())
epsilon = 1e-3
found = False
for i in range(5, len(Jclust_values) + 4):
    if Jclust_values[i - 5] - Jclust_values[i - 4] <= epsilon:
        print(f'+++++ Best number of clusters could be considered as: {i}')
        found = True
        break

if not found:
    print(f'+++++ Best number of clusters could be considered as:
↪{len(Jclust_values) + 4}')
```

```
+++++ Best number of clusters could be considered as: 20
```

- The choice of initial condition have some effects on the performance of k-means clustering algorithm in terms of **faster convergence** (consider luckily `initializing with the converged centroids`), and **better clusters** (i.e. `lower Jclust values`)