**Q1.** (40 points) You are given the following dataset.

| | $I_1$ | $I_2$ | T |
|---|---|---|---|
| $e_1$ | 2 | 1 | 0 |
| $e_2$ | -1 | 3 | 1 |
| $e_3$ | 5 | 1 | 0 |
| $e_4$ | 1 | 3 | 1 |

Run the perceptron algorithm (p19 of chap3e_ann1x.pdf) by hand, and write the results in the following format.

| iteration | $W^{old}$ | I | T | O | T = 0? | $W^{new}$ |
|---|---|---|---|---|---|---|
| 0 | [0 0 0] | [2 1] | 0 | 0 | yes | [0 0 0] |
| 1 | [0 0 0] | [-1 3] | 0 | 1 | no | [1 -1 3] |
| 2 | [1 -1 3] | [5 1] | 0 | 0 | yes | [1 -1 3] |
| 3 | [1 -1 3] | [1 3] | 1 | 1 | yes | [1 -1 3] |
| 4 | [1 -1 3] | [2 1] | 1 | 0 | no | [0 -3 2] |
| 5 | [0 -3 2] | [-1 3] | 1 | 1 | yes | [0 -3 2] |
| 6 | [0 -3 2] | [5 1] | 0 | 0 | yes | [0 -3 2] |
| 7 | [0 -3 2] | [1 3] | 1 | 1 | yes | [0 -3 2] |

# q2

November 14, 2019

## 0.1 Q2a. (40 points) In this question, you are required to implement (using Keras) a 10-output CNN with the following layers:

1. 16 channels of $2 \times 2$ convolution, with ReLU activation;
2. max-pooling layer with stride 2;
3. 16 channels of $2 \times 2$ convolution, with ReLU activation;
4. max-pooling layer with stride 2;
5. fully connected layer with 120 ReLU hidden units;
6. another fully connected layer with 64 ReLU hidden units.

In this experiment, we use the MNIST (https://hkustconnect-my.sharepoint.com/:u:/g/personal/hzhangal_connect_ust_hk/ERzaZJwExepPlf92u_1cCPABLyuC21lBggcZ9GHx0mpyPQ?e=YklceJ) dataset. The first column is the class label. The other columns are the intensity values for each individual pixel in each MNIST image. Each student will have his/her own test set (which is based on your student id). Run your code using adam as the optimizer, train your model for 10 epochs on the training set, and report the accuracy on your test set.

[1]:
```
import numpy as np
import keras
import pandas as pd
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
np.random.seed(0)
from keras import backend as K
```

```
Using TensorFlow backend.
```

[16]:
```
#loading data
train = np.genfromtxt('asgn2_data/train.csv', delimiter = ',')
test = np.genfromtxt('asgn2_data/20380937.csv', delimiter = ',')
display(train.shape)
```

```
(1581, 785)
```

[26]:
```
x_train = train[1:,1:].reshape((-1, 28, 28, 1))
x_test = test[1:,1:].reshape((-1, 28, 28, 1))
y_train = train[1:, 0]
```

```
y_test = test[1:,0]
```

[ ]:
```
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')

# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, 10)
y_test = keras.utils.to_categorical(y_test, 10)
```

[27]:
```
# Building the model
"""
1. 16 channels of 2 × 2 convolution, with ReLU activation; 2. max-pooling layer␣
↪with stride 2;
3. 16 channels of 2 × 2 convolution, with ReLU activation;
1
4. max-pooling layer with stride 2;
5. fully connected layer with 120 ReLU hidden units;
6. another fully connected layer with 64 ReLU hidden units"""
model = Sequential()

model.add(Conv2D(16, kernel_size=(2, 2), activation='relu', input_shape=(28,␣
↪28, 1)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(16, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())

model.add(Dense(120, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(10, activation='softmax'))
```

[28]:
```
# training
model.compile(loss=keras.losses.sparse_categorical_crossentropy,
 optimizer=keras.optimizers.Adam(),
 metrics=['accuracy'])
```

[29]:
```
model.fit(x_train, y_train,
          batch_size=128,
          epochs=10,
          verbose=1,
          validation_data=(x_test, y_test))
```

```
Train on 1580 samples, validate on 210 samples
Epoch 1/10
1580/1580 [==============================] - 1s 588us/step - loss: 2.0321 -
accuracy: 0.3994 - val_loss: 1.5831 - val_accuracy: 0.6714
```

```
Epoch 2/10
1580/1580 [==============================] - 0s 272us/step - loss: 1.1609 -
accuracy: 0.7291 - val_loss: 0.7597 - val_accuracy: 0.7905
Epoch 3/10
1580/1580 [==============================] - 0s 291us/step - loss: 0.6039 -
accuracy: 0.8228 - val_loss: 0.5039 - val_accuracy: 0.8619
Epoch 4/10
1580/1580 [==============================] - 0s 279us/step - loss: 0.4219 -
accuracy: 0.8772 - val_loss: 0.4423 - val_accuracy: 0.8762
Epoch 5/10
1580/1580 [==============================] - 0s 285us/step - loss: 0.3257 -
accuracy: 0.9038 - val_loss: 0.3181 - val_accuracy: 0.9143
Epoch 6/10
1580/1580 [==============================] - 1s 352us/step - loss: 0.2637 -
accuracy: 0.9259 - val_loss: 0.2813 - val_accuracy: 0.9238
Epoch 7/10
1580/1580 [==============================] - 0s 273us/step - loss: 0.2224 -
accuracy: 0.9361 - val_loss: 0.2579 - val_accuracy: 0.9238
Epoch 8/10
1580/1580 [==============================] - 0s 285us/step - loss: 0.1874 -
accuracy: 0.9494 - val_loss: 0.2659 - val_accuracy: 0.9190
Epoch 9/10
1580/1580 [==============================] - 0s 280us/step - loss: 0.1713 -
accuracy: 0.9525 - val_loss: 0.2240 - val_accuracy: 0.9429
Epoch 10/10
1580/1580 [==============================] - 0s 281us/step - loss: 0.1397 -
accuracy: 0.9652 - val_loss: 0.1883 - val_accuracy: 0.9429
```

[29]:
```
<keras.callbacks.callbacks.History at 0x1350e6748>
```

[30]:
```python
#testing
loss, acc = model.evaluate(x_test, y_test, verbose=0)
print("accuracy for the given CNN architecture: ", acc)
```

```
accuracy for the given CNN architecture:  0.9428571462631226
```

#Changing parameters for part b

[34]:
```python
# model
model = Sequential()
model.add(Conv2D(16, kernel_size=(4, 4), activation='relu',input_shape=(28, 28,␣
↪1))) # unknown model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(16, (4, 4), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(120, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(10, activation='softmax'))
```

[35]:
```
model.compile(loss=keras.losses.sparse_categorical_crossentropy,␣
 ↪optimizer=keras.optimizers.Adam(),
metrics=['accuracy'])
```

[36]:
```
model.fit(x_train, y_train, epochs=10,
verbose=2, shuffle=False)
```

```
Epoch 1/10
 - 2s - loss: 1.0619 - accuracy: 0.6804
Epoch 2/10
 - 2s - loss: 0.3163 - accuracy: 0.9032
Epoch 3/10
 - 2s - loss: 0.1680 - accuracy: 0.9532
Epoch 4/10
 - 2s - loss: 0.1090 - accuracy: 0.9633
Epoch 5/10
 - 2s - loss: 0.0660 - accuracy: 0.9791
Epoch 6/10
 - 2s - loss: 0.0441 - accuracy: 0.9861
Epoch 7/10
 - 1s - loss: 0.0301 - accuracy: 0.9943
Epoch 8/10
 - 2s - loss: 0.0242 - accuracy: 0.9930
Epoch 9/10
 - 1s - loss: 0.0179 - accuracy: 0.9949
Epoch 10/10
 - 1s - loss: 0.0099 - accuracy: 0.9962
```

[36]:
```
<keras.callbacks.callbacks.History at 0x1360d5f60>
```

[39]:
```
loss, acr = model.evaluate(x_test, y_test)
print("accuracy for model in part b after making changes: ", acr)
```

```
210/210 [==============================] - 0s 364us/step
accuracy for model in part b after making changes:  0.9523809552192688
```

# q3

November 14, 2019

## 0.1 Q3. (20 points) In this question, you use the code segments provided in the tutorial to implement the k-means and k-medoid algorithms as follows:

1. kmeans k input-file output-file
2. kmedoids k input-file output-file

k-means and k-medoid algorithms as follows: The input-file is the dataset (with one sample per line), and the output-output is a png file, which use different colors to show the different clusters. An example is shown below.

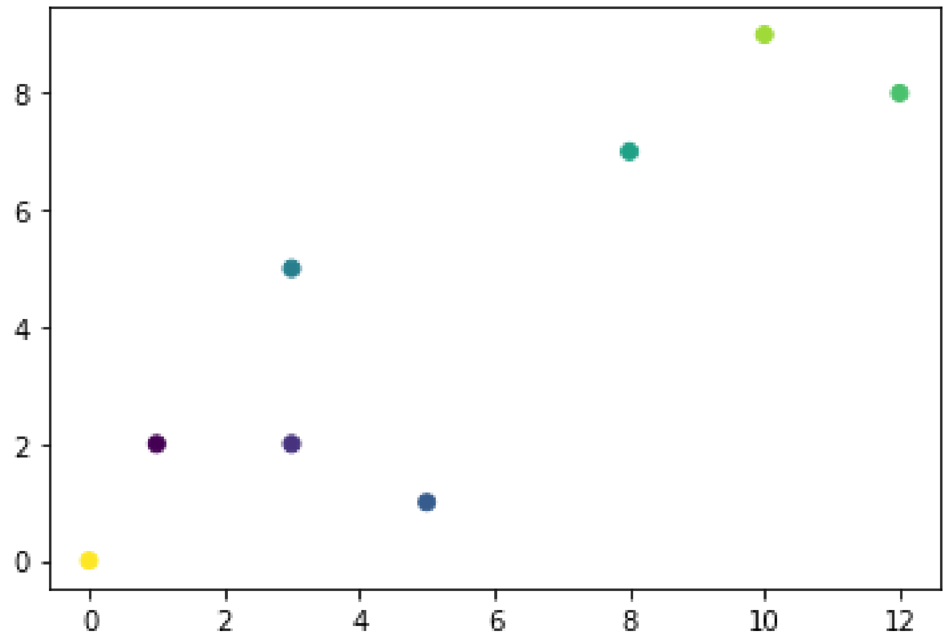
**Please find the code inside "q2.py" that uses sys argv[] to run the kmeans and kmediod in the folder. This is the visual representation used for solution**

[2]:
```
# Copying code segment from Lab 5
from sklearn.datasets import make_blobs
from matplotlib import pyplot
import numpy as np
import random
from matplotlib import pyplot as plt
from scipy.cluster.hierarchy import dendrogram, linkage


class Clustering():
    def __init__(self, k_num_center = 2, outlier = True, method = 'Mean'):
        self.k_num_center = k_num_center
        self.method = method
        if outlier:
            self.data = np.
↪array([[1,2],[3,2],[5,1],[3,5],[8,7],[12,8],[10,9],[0,0],[40,40]])
            target = np.array([1,2,3,4,5,6,7,8,9])
        else:
            self.data = np.
↪array([[1,2],[3,2],[5,1],[3,5],[8,7],[12,8],[10,9],[0,0]])
            target = np.array([1,2,3,4,5,6,7,8])
        pyplot.scatter(self.data[:,0],self.data[:,1], c = target)
        pyplot.show()

    def ou_distance(self,x,y):
        return np.sqrt(sum(np.square(x-y)))
```

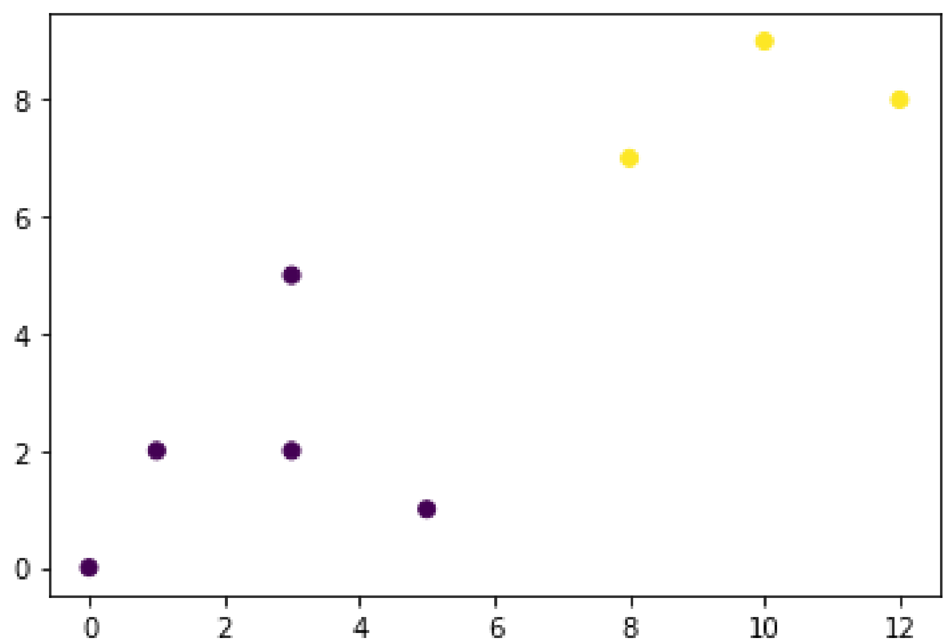
```python
    def run_k_means(self, func_of_dis):
        indexs = list(range(len(self.data)))
        random.shuffle(indexs)
        init_centroids_index = indexs[:self.k_num_center]
        centroids = self.data[init_centroids_index,:]
        levels = list(range(self.k_num_center))
        print("start iteration")
        sample_target = []
        for i in range(10):
            new_centroids = [[] for i in range(self.k_num_center)]
            new_centroids_num = [0 for i in range(self.k_num_center)]
            sample_target = []

            for sample in self.data:
                distances = [self.ou_distance(sample, centroid) for centroid in␣
↪centroids]
                cur_level = np.argmin(distances)
                sample_target.append(cur_level)


                new_centroids_num[cur_level]+=1
                if len(new_centroids[cur_level]) < 1:
                    new_centroids[cur_level] = sample
                else:
                    new_centroids[cur_level] = new_centroids[cur_level]+sample

            centroids = list()
            for centroid, num in zip(new_centroids, new_centroids_num):
                centroids.append([item/num for item in centroid])
            centroids = np.array(centroids)

        print('end')
        return sample_target

    def run_k_center(self, func_of_dis):
        print('randomly create', self.k_num_center, 'centers')
        indexs = list(range(len(self.data)))
        random.shuffle(indexs)
        init_centroids_index = indexs[:self.k_num_center]
        centroids = self.data[init_centroids_index,:]
        levels = list(range(self.k_num_center))
        print("start iteration")
        sample_target = []
        if_stop = False
        while(not if_stop):
            if_stop = True
```

```
            classify_points = [[centroid] for centroid in centroids]
            sample_target = []

            for sample in self.data:
                distances = [func_of_dis(sample, centroid) for centroid in␣
↪centroids]
                cur_level = np.argmin(distances)
                sample_target.append(cur_level)
                classify_points[cur_level].append(sample)

            for i in range(self.k_num_center):
                distances = [func_of_dis(point_1, centroids[i]) for point_1 in␣
↪classify_points[i]]
                now_distances = sum(distances)
                for point in classify_points[i]:
                    distances = [func_of_dis(point_1, point) for point_1 in␣
↪classify_points[i]]
                    new_distance = sum(distances)
                    if new_distance < now_distances:
                        now_distances = new_distance
                        centroids[i] = point
                        if_stop = False

        print('end')
        return sample_target

    def run(self):
        if self.method == 'Mean':
            predict = self.run_k_means(self.ou_distance)
        else:
            predict = self.run_k_center(self.ou_distance)
        pyplot.scatter(self.data[:,0], self.data[:,1], c=predict)
```

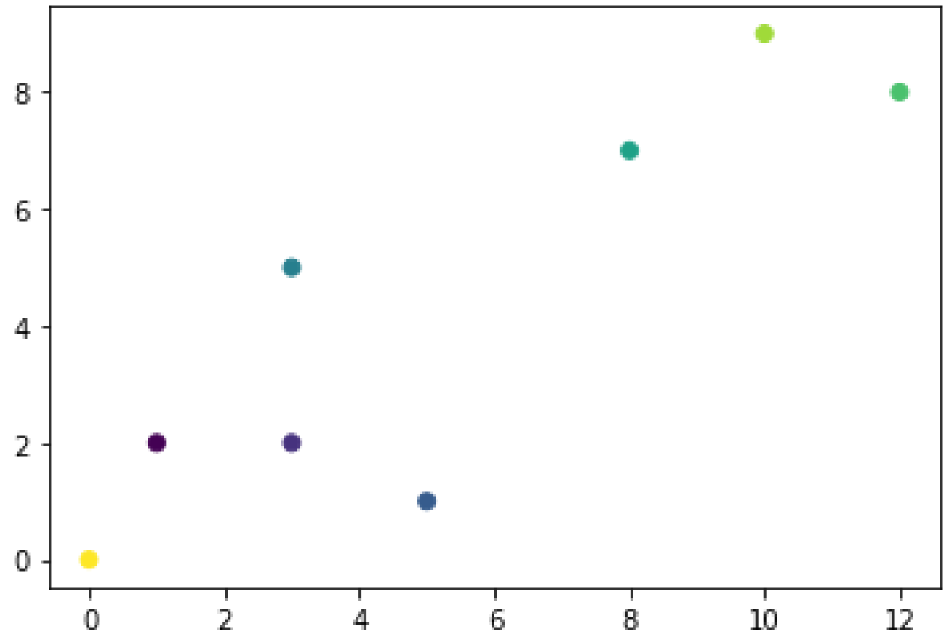
[3]:
```
test_one = Clustering(outlier = False, k_num_center = 2, method = 'Mean')
test_one.run()
```



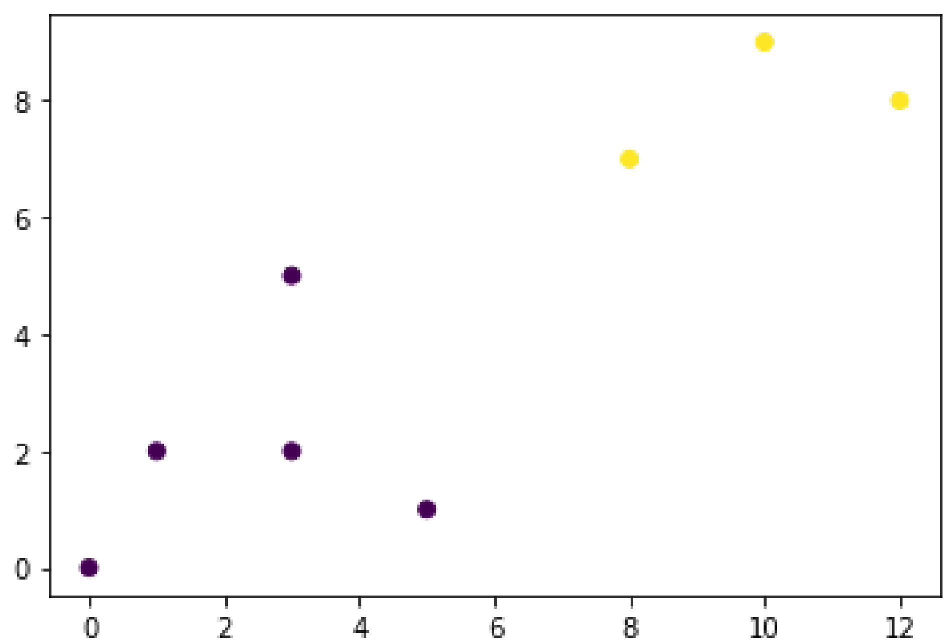

```
start iteration
end
```



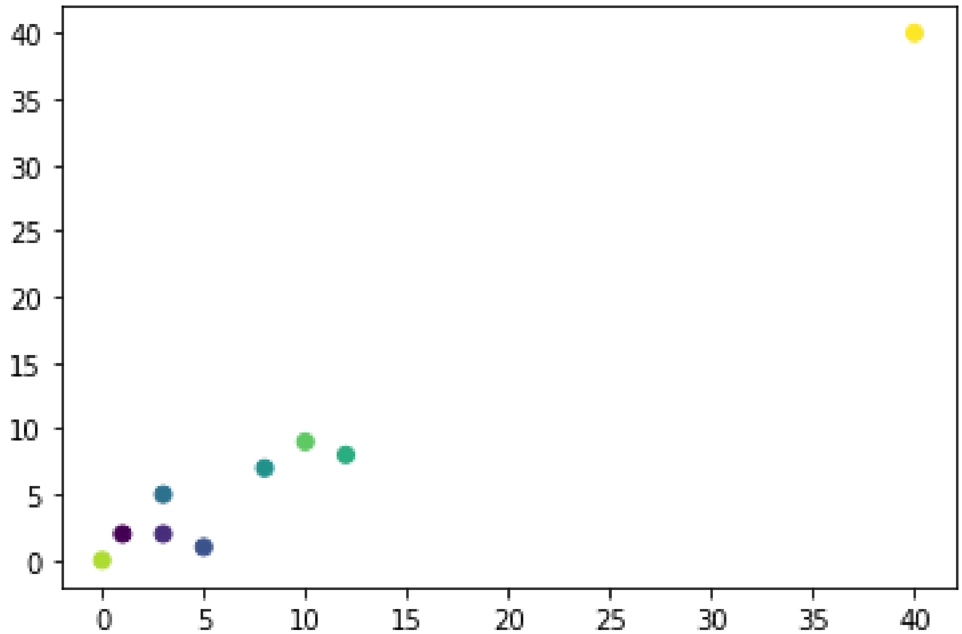
[5]:
```
test_one = Clustering(outlier = False, k_num_center = 2, method = 'medoid')
test_one.run()
```



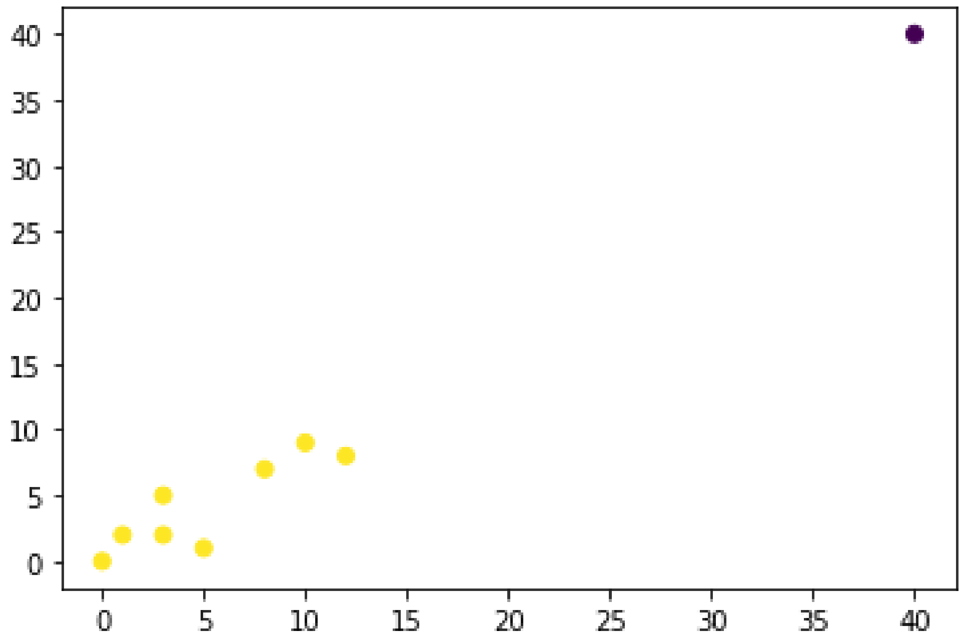

```
randomly create 2 centers
start iteration
end
```



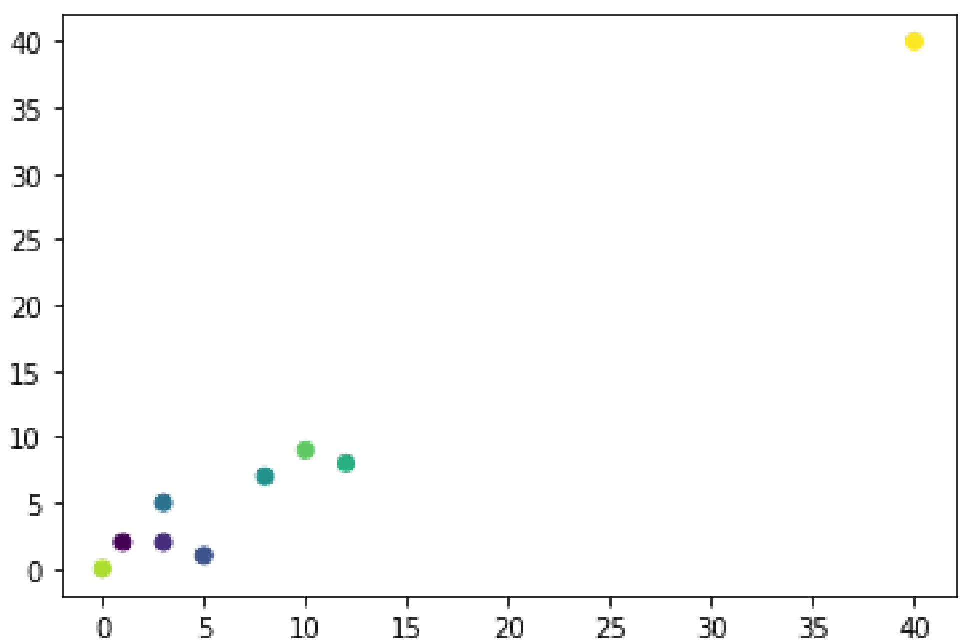
[6]:
```
test_one = Clustering(outlier = True, k_num_center = 2, method = 'Mean')
test_one.run()
```



```
start iteration
end
```

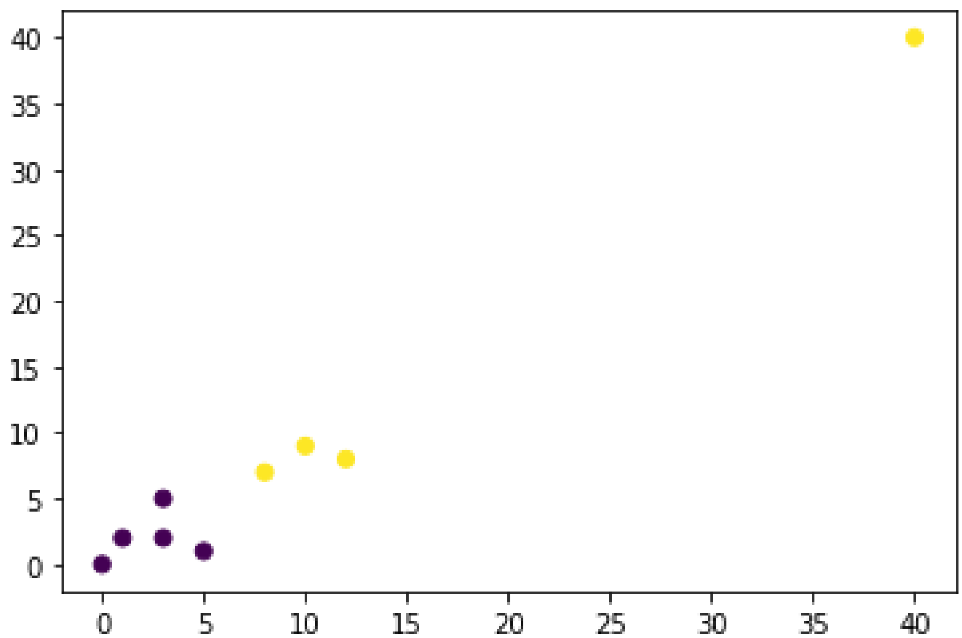



```
[7]: test_one = Clustering(outlier = True, k_num_center = 2, method = 'medoid')
     test_one.run()
```




```
randomly create 2 centers
start iteration
```

```
end
```




c. We can see that kmediod accounts for the weight of the outliers in creating clusters and are not much impacted in the presence of outliers in the data. On the otherhand, k means is heavily affected by the presence of outliers but works just as well when outliers = False

[ ]:

```

```