University of Minho
School of Engineering

ISLab

Dados e Aprendizagem Automática

Unsupervised Learning:

K-means and K-medoids

DAA @ MEI-1º/MiEI-4º – 1º Semestre
Bruno Fernandes, César Analide, Dalila Alves, Filipa Ferraz, Victor Alves

Part VII

Contents

- Unsupervised Learning
  - K-Means Clustering
  - K-Medoids Clustering
- Hands On

Unsupervised Learning

Unsupervised Learning - K-Means and K-Medoids

- Exercise:
  - **Problem:** Development of a Machine Learning Model able to *cluster data based on their similarity*
  - **Classification Approach:** Clustering algorithms to solve this problem

Method Used

- Unsupervised learning means that there is no outcome to be predicted, and the algorithm just tries to find patterns in the data.

  → tries to cluster data based on their similarity.

  - **Dataset:** *Generate isotropic Gaussian blobs* dataset for clustering using *sklearn.datasets.make_blobs*

  - **Note:** to apply K-Medoids Clustering it is required to install package *scikit-learn-extra*

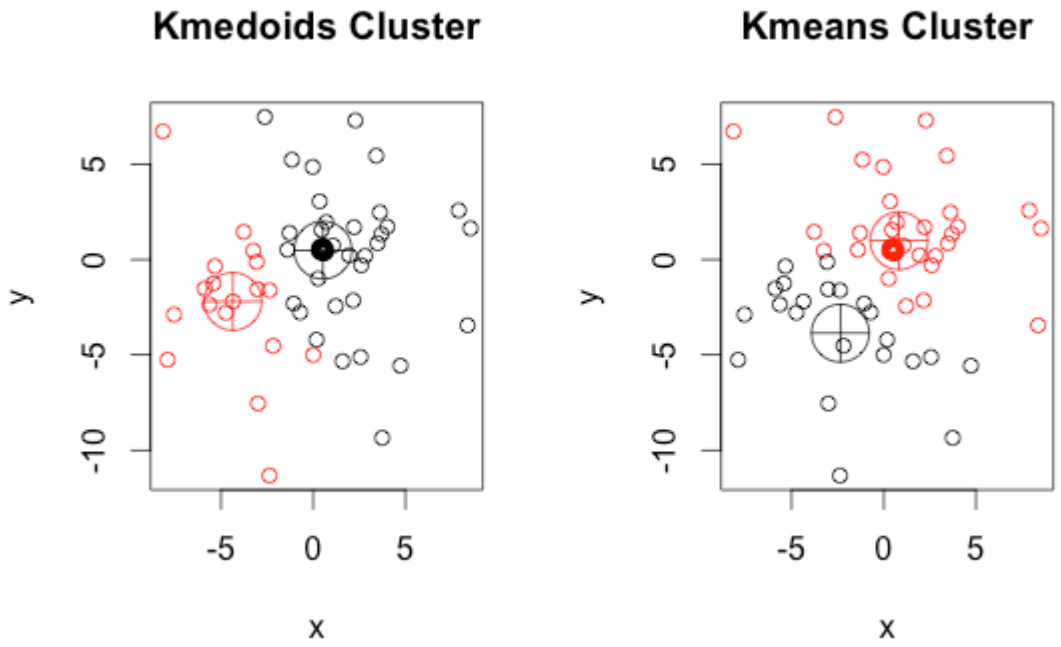
```
conda install -c conda-forge scikit-learn-extra
```

or

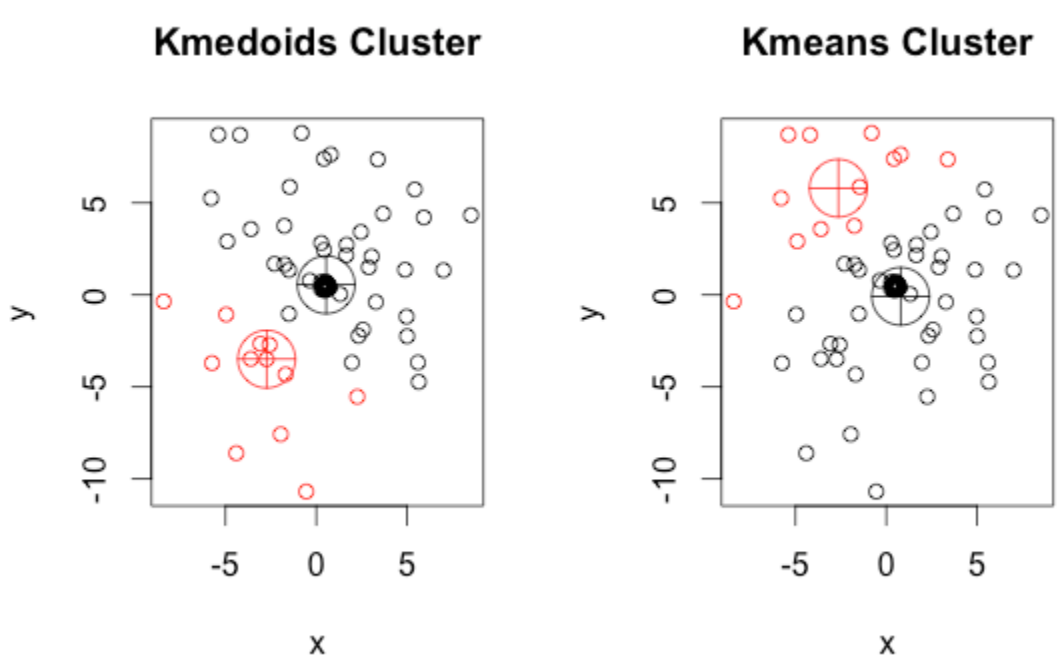
```
pip install scikit-learn-extra
```

Unsupervised Learning - K-Means and K-Medoids

1st iteration



2nd iteration

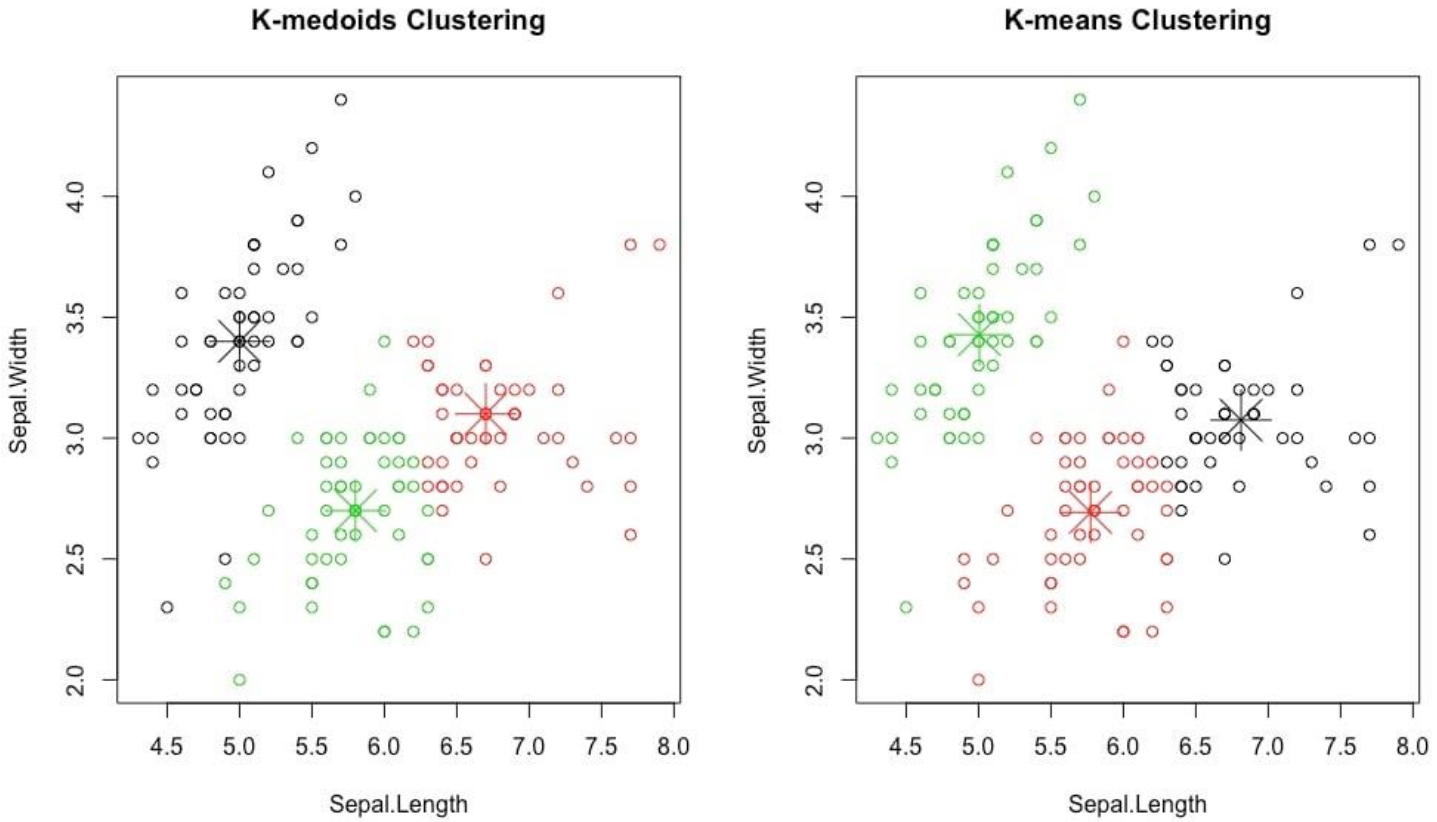


https://stats.stackexchange.com/questions/156210/an-example-where-the-output-of-the-k-medoid-algorithm-is-different-than-the-outp

Unsupervised Learning - K-Means and K-Medoids

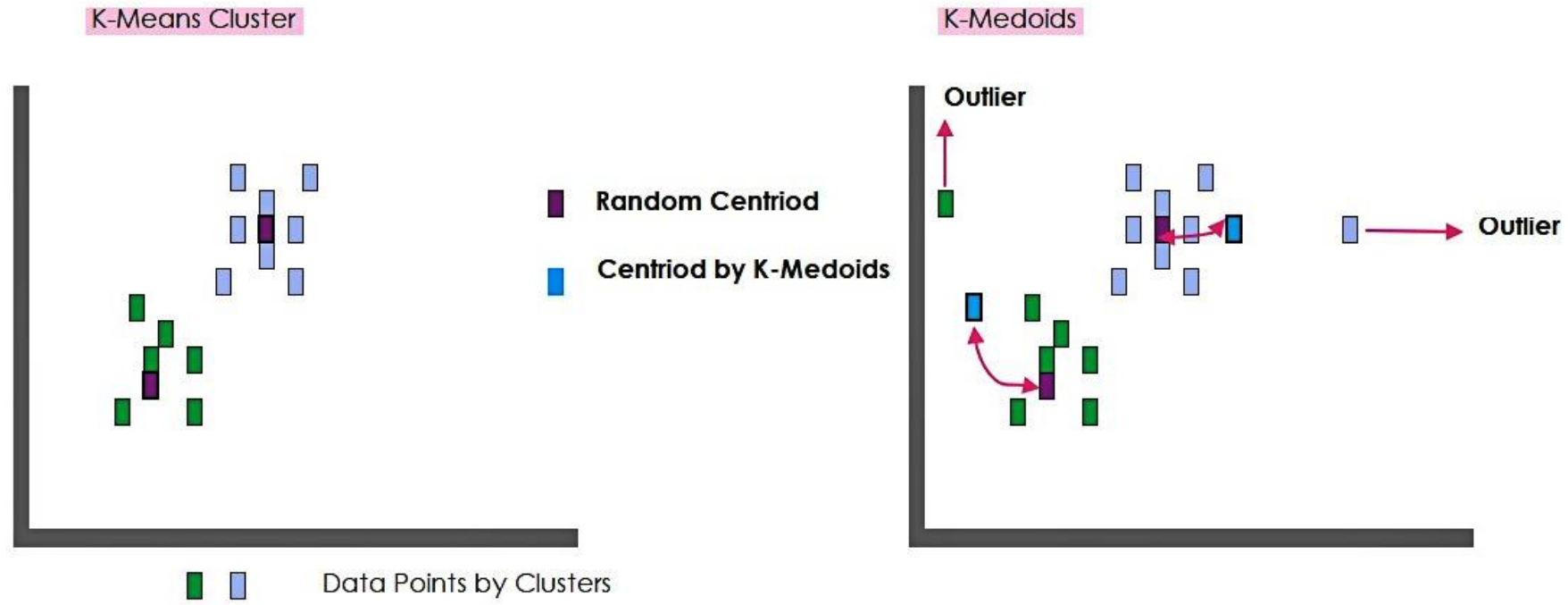
- Both K-Means and K-Medoids algorithms are:
  - breaking the dataset up into **k groups**;
  - trying to **minimize the distance between points of the same cluster and a particular point** which is the centre of that cluster.
- In contrast to the K-Means algorithm, the K-Medoids algorithm **chooses points as centres that belong to the dataset**.
- The **most common implementation** of the K-Medoids clustering algorithm is the **Partitioning Around Medoids (PAM)** algorithm.
- The **PAM** algorithm uses **a greedy search**, which may not find the global optimum solution.
- **Medoids are more robust to outliers than** centroids, but they **need more computation** for high dimensional data.

Unsupervised Learning - K-Means and K-Medoids



https://subscription.packtpub.com/book/big-data-and-business-intelligence/9781789956399/1/ch01lvl1sec08/introduction-to-k-medoids-clustering

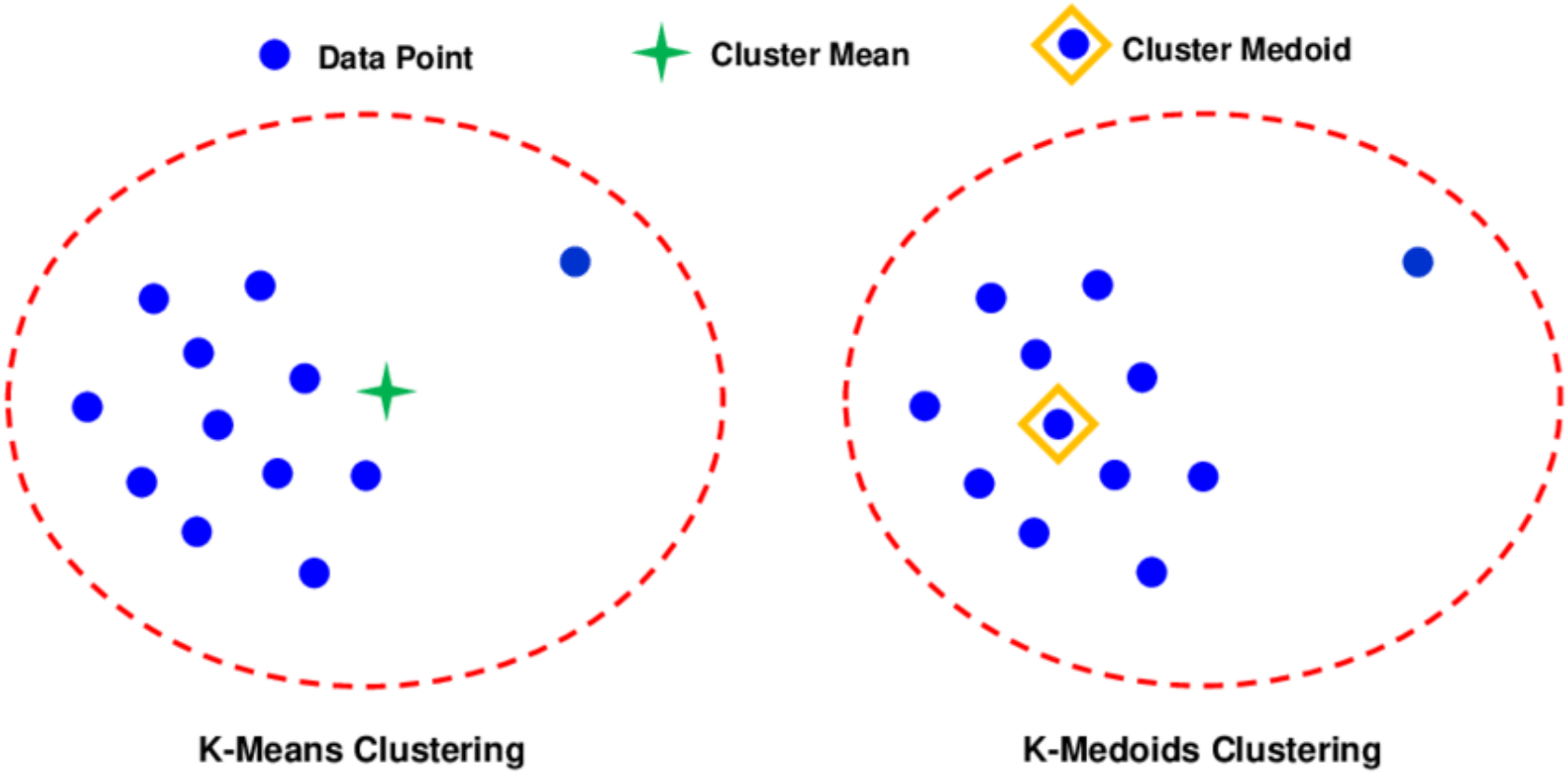
Unsupervised Learning - K-Means and K-Medoids



The Choice of Centriod in K-Means is random and what if there is presence of Outlier?
K-Medoids solves the issue as it K-Medoids selected the precise centroid among all data points of the corresponding clusters

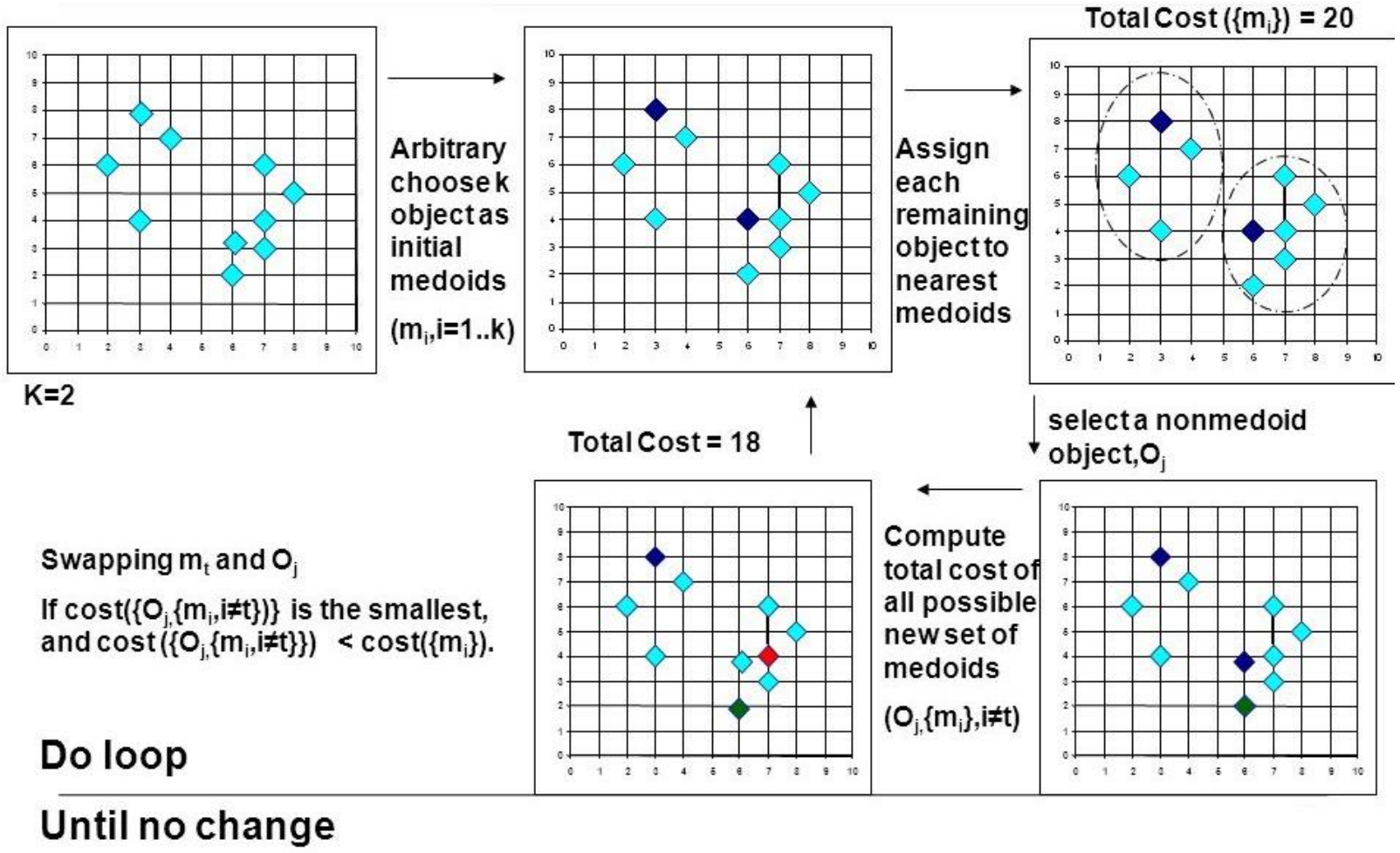
https://www.kaggle.com/getting-started/172928

Unsupervised Learning - K-Means and K-Medoids



https://www.researchgate.net/figure/The-graphical-representation-of-the-difference-between-the-k-means-and-k-medoids_fig1_342871651

Unsupervised Learning - K-Means and K-Medoids



https://alfredo-reyes-montero.gitbook.io/soft-computing/applications/matlab/kmeans-kmenoids

Unsupervised Learning - K-Means and K-Medoids

Import libraries

```
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

Create the data

```
from sklearn.datasets import make_blobs
```

```
data = make_blobs(n_samples = 200, n_features = 2, centers = 4,
                  cluster_std = 1.8, random_state = 2022)
```

```
X = data[0]
y = data[1]
```

View the first 5 lines of the data set:

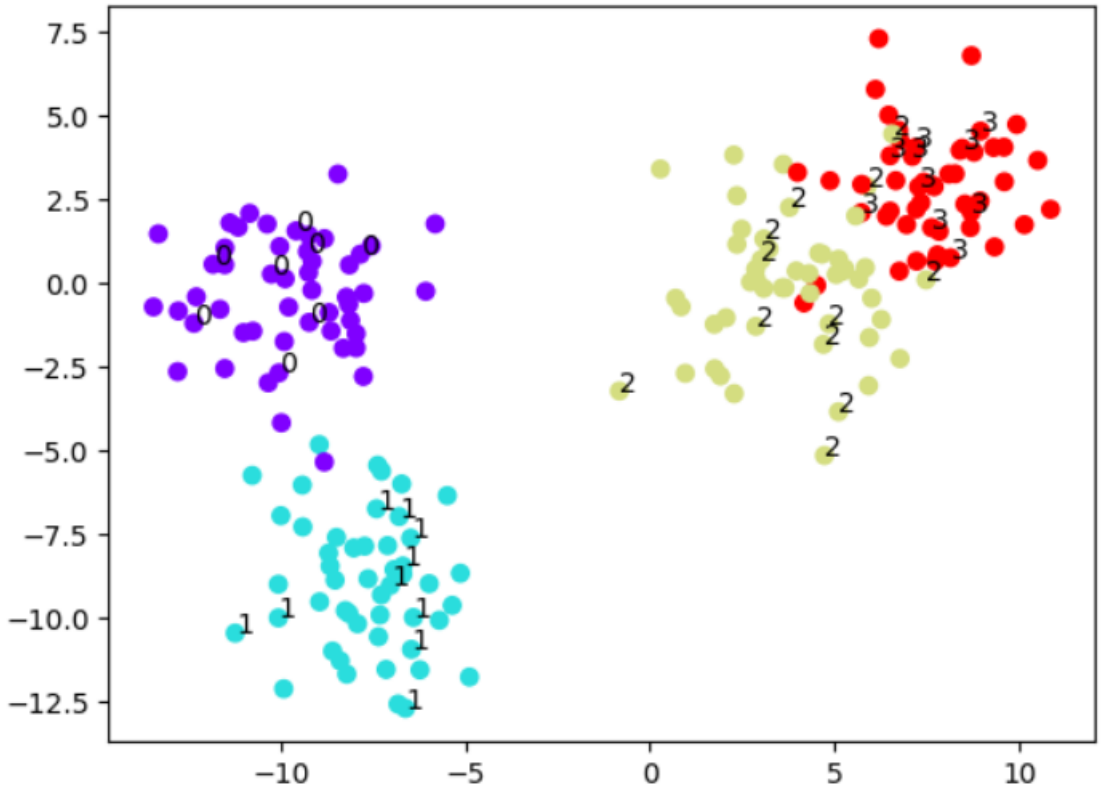
```
print('X:', X[0:5, :])
print('Y:', y[0:5])
```

```
X: [[  5.88508997   2.9021639 ]
 [ -8.20429992 -11.68670283]
 [  1.9125188   -2.76746603]
 [ -9.39601207  -7.2830252 ]
 [  6.1986976    7.32152342]]
Y: [2 1 2 1 3]
```

4 blobs

Visualize the data

```
plt.scatter(X[:, 0], X[:, 1], c = y, cmap = 'rainbow')
for i, txt in enumerate(y):
    if i%5 == 0:
        plt.annotate(txt, (X[i, 0], X[i, 1]))
```



Unsupervised Learning - K-Means

Create the clusters

With K-Means

```
from sklearn.cluster import KMeans
```

```
kmeans = KMeans(n_clusters = 4, n_init = 10, random_state = 2022)
```

```
kmeans.fit(X)
```

```
KMeans(n_clusters=4, n_init=10, random_state=2022)
```

```
kmeans.cluster_centers_
```

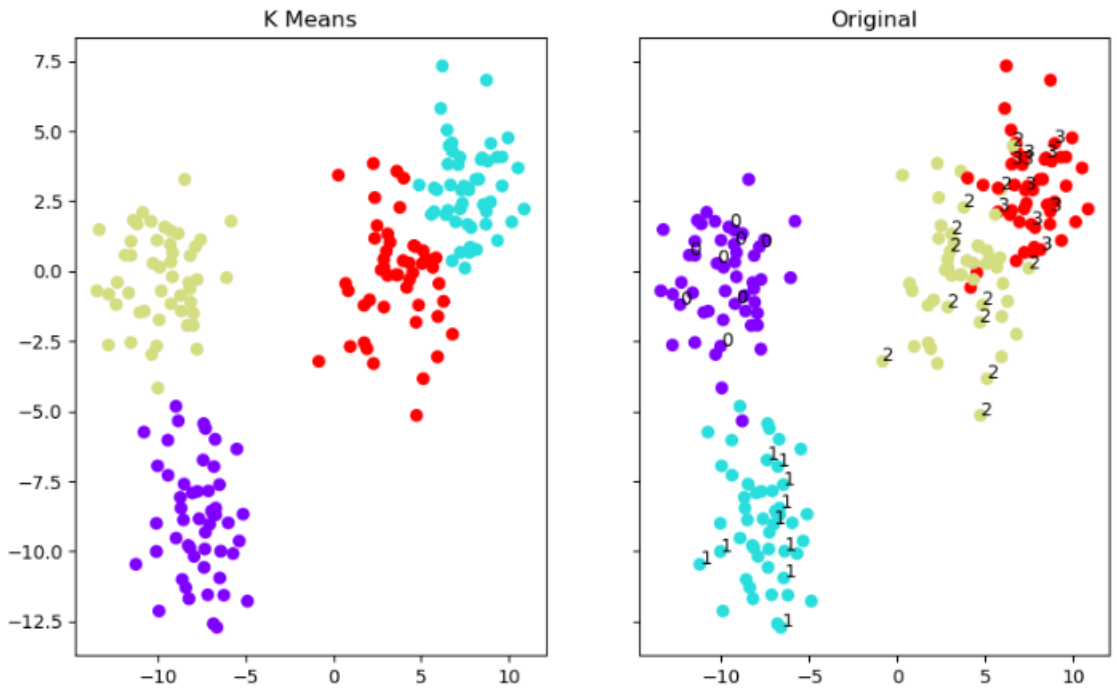
```
array([[-7.68797564, -8.88054369],
       [ 7.70499062,  2.96975295],
       [-9.78544862, -0.2509739 ],
       [ 3.60428123, -0.21752545]])
```

```
kmeans.labels_
```

```
array([1, 0, 3, 0, 1, 2, 0, 2, 3, 3, 1, 1, 3, 1, 2, 1, 1, 1, 1, 2, 0, 3,
       1, 1, 0, 0, 3, 0, 3, 3, 1, 0, 1, 3, 1, 3, 0, 1, 0, 2, 1, 2, 1, 3,
       3, 3, 0, 0, 3, 0, 0, 3, 1, 1, 0, 3, 1, 3, 1, 2, 2, 2, 3, 3, 1, 2,
       0, 2, 2, 2, 3, 2, 3, 0, 0, 2, 2, 0, 0, 2, 1, 1, 2, 3, 0, 0, 1, 3,
       2, 0, 3, 1, 2, 3, 3, 1, 3, 2, 3, 3, 0, 1, 2, 0, 2, 3, 2, 3, 2, 2,
       1, 2, 2, 2, 3, 2, 3, 3, 0, 3, 1, 1, 1, 3, 0, 0, 1, 3, 2, 1, 0, 2,
       2, 0, 0, 0, 2, 2, 1, 0, 0, 0, 1, 3, 0, 1, 0, 2, 3, 2, 3, 3, 1, 1,
       1, 1, 1, 1, 0, 0, 0, 0, 0, 3, 1, 2, 0, 0, 1, 2, 3, 1, 1, 0, 3, 1,
       3, 0, 0, 2, 3, 0, 2, 2, 0, 2, 1, 1, 2, 3, 2, 3, 3, 0, 2, 1, 2, 0,
       2, 2])
```

Unsupervised Learning - K-Means


```
f, (ax1, ax2) = plt.subplots(1, 2, sharey = True, figsize = (10, 6))
ax1.set_title('K Means')
ax1.scatter(X[:, 0], X[:, 1], c = kmeans.labels_, cmap = 'rainbow')
ax2.set_title("Original")
ax2.scatter(X[:, 0], X[:, 1], c = y, cmap = 'rainbow')
for i, txt in enumerate(y):
    if i%5 == 0:
        plt.annotate(txt, (X[i, 0], X[i, 1]))
```



You should note that the colors are meaningless when in reference to the two plots.

Unsupervised Learning - K-Means

Align K-Means prediction class with real values

```
from sklearn.metrics import classification_report, confusion_matrix
import numpy as np
```

```
y_pred = kmeans.predict(X)
```

```
y_pred
```

```
array([1, 0, 3, 0, 1, 2, 0, 2, 3, 3, 1, 1, 3, 1, 2, 1, 1, 1, 1, 2, 0, 3,
       1, 1, 0, 0, 3, 0, 3, 3, 1, 0, 1, 3, 1, 3, 0, 1, 0, 2, 1, 2, 1, 3,
       3, 3, 0, 0, 3, 0, 0, 3, 1, 1, 0, 3, 1, 3, 1, 2, 2, 2, 3, 3, 1, 2,
       0, 2, 2, 2, 3, 2, 3, 0, 0, 2, 2, 0, 0, 2, 1, 1, 2, 3, 0, 0, 1, 3,
       2, 0, 3, 1, 2, 3, 3, 1, 3, 2, 3, 3, 0, 1, 2, 0, 2, 3, 2, 3, 2, 2,
       1, 2, 2, 2, 3, 2, 3, 3, 0, 3, 1, 1, 1, 3, 0, 0, 1, 3, 2, 1, 0, 2,
       2, 0, 0, 0, 2, 2, 1, 0, 0, 0, 1, 3, 0, 1, 0, 2, 3, 2, 3, 3, 1, 1,
       1, 1, 1, 1, 0, 0, 0, 0, 0, 3, 1, 2, 0, 0, 1, 2, 3, 1, 1, 0, 3, 1,
       3, 0, 0, 2, 3, 0, 2, 2, 0, 2, 1, 1, 2, 3, 2, 3, 3, 0, 2, 1, 2, 0,
       2, 2])
```

```
y
```

```
array([2, 1, 2, 1, 3, 0, 1, 0, 2, 2, 3, 3, 2, 3, 0, 3, 3, 3, 3, 0, 1, 2,
       3, 3, 1, 1, 3, 1, 2, 2, 3, 1, 3, 3, 3, 2, 1, 3, 1, 0, 3, 0, 3, 2,
       2, 2, 1, 1, 2, 1, 1, 2, 3, 3, 1, 2, 3, 2, 3, 0, 0, 0, 2, 2, 3, 0,
       1, 0, 0, 0, 2, 0, 2, 1, 1, 0, 0, 1, 1, 0, 2, 3, 0, 2, 1, 1, 3, 2,
       0, 1, 2, 3, 0, 2, 2, 3, 2, 0, 2, 2, 1, 3, 0, 1, 0, 2, 0, 2, 0, 0,
       3, 0, 0, 0, 2, 0, 2, 2, 1, 2, 3, 3, 3, 2, 1, 1, 3, 2, 0, 3, 1, 0,
       0, 1, 1, 1, 0, 0, 3, 1, 1, 1, 3, 2, 1, 3, 1, 0, 2, 0, 2, 2, 3, 3,
       2, 3, 3, 3, 0, 1, 1, 1, 1, 2, 3, 0, 1, 1, 3, 0, 2, 3, 3, 1, 2, 3,
       2, 1, 1, 0, 2, 1, 0, 0, 1, 0, 3, 3, 0, 2, 0, 2, 2, 1, 0, 3, 0, 1,
       0, 0])
```

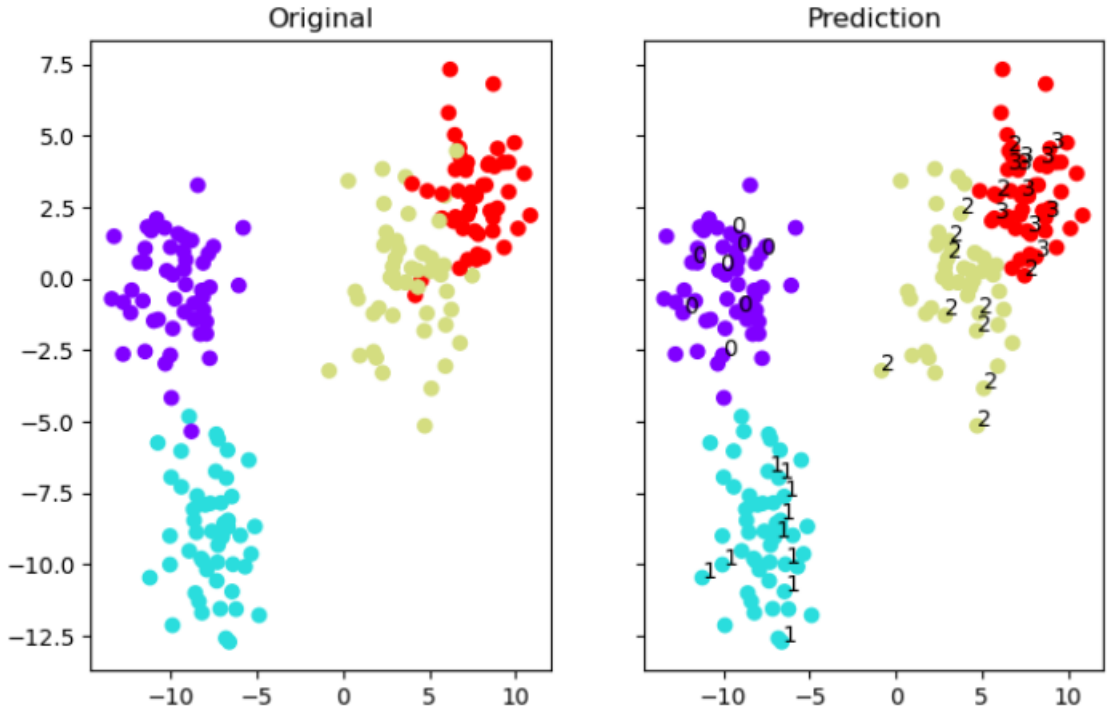
```
y_pred = np.where(y_pred==0, 10, y_pred)
y_pred = np.where(y_pred==2, 0, y_pred)
y_pred = np.where(y_pred==3, 2, y_pred)
y_pred = np.where(y_pred==1, 3, y_pred)
y_pred = np.where(y_pred==10, 1, y_pred)
```

```
y_pred
```

```
array([3, 1, 2, 1, 3, 0, 1, 0, 2, 2, 3, 3, 2, 3, 0, 3, 3, 3, 3, 0, 1, 2,
       3, 3, 1, 1, 2, 1, 2, 2, 3, 1, 3, 2, 3, 2, 1, 3, 1, 0, 3, 0, 3, 2,
       2, 2, 1, 1, 2, 1, 1, 2, 3, 3, 1, 2, 3, 2, 3, 0, 0, 0, 2, 2, 3, 0,
       1, 0, 0, 0, 2, 0, 2, 1, 1, 0, 0, 1, 1, 0, 3, 3, 0, 2, 1, 1, 3, 2,
       0, 1, 2, 3, 0, 2, 2, 3, 2, 0, 2, 2, 1, 3, 0, 1, 0, 2, 0, 2, 0, 0,
       3, 0, 0, 0, 2, 0, 2, 2, 1, 2, 3, 3, 3, 2, 1, 1, 3, 2, 0, 3, 1, 0,
       0, 1, 1, 1, 0, 0, 3, 1, 1, 1, 3, 2, 1, 3, 1, 0, 2, 0, 2, 2, 3, 3,
       3, 3, 3, 3, 1, 1, 1, 1, 1, 2, 3, 0, 1, 1, 3, 0, 2, 3, 3, 1, 2, 3,
       2, 1, 1, 0, 2, 1, 0, 0, 1, 0, 3, 3, 0, 2, 0, 2, 2, 1, 0, 3, 0, 1,
       0, 0])
```

Unsupervised Learning - K-Means


```
f, (ax1, ax2) = plt.subplots(1, 2, sharey = True, figsize = (8, 5))
ax1.set_title('Original')
ax1.scatter(X[:,0], X[:,1], c=y, cmap='rainbow')
ax2.set_title("Prediction")
ax2.scatter(X[:,0], X[:,1], c=y_pred, cmap='rainbow')
for i, txt in enumerate(y):
    if i%5 == 0:
        plt.annotate(txt, (X[i,0], X[i,1]))
plt.savefig("KMeans_pred.png")
```




```
print(confusion_matrix(y, y_pred))
```

```
[[49  1  0  0]
 [ 0 50  0  0]
 [ 0  0 46  4]
 [ 0  0  3 47]]
```

```
print(classification_report(y, y_pred))
```

	precision	recall	f1-score	support
0	1.00	0.98	0.99	50
1	0.98	1.00	0.99	50
2	0.94	0.92	0.93	50
3	0.92	0.94	0.93	50
accuracy			0.96	200
macro avg	0.96	0.96	0.96	200
weighted avg	0.96	0.96	0.96	200

Unsupervised Learning - K-Medoids

With K-Medoids

```
from sklearn_extra.cluster import KMedoids
```

```
kmedoids = KMedoids(n_clusters=4, random_state=2022)
kmedoids.fit(X)
```

KMedoids

KMedoids(n_clusters=4, random_state=2022)

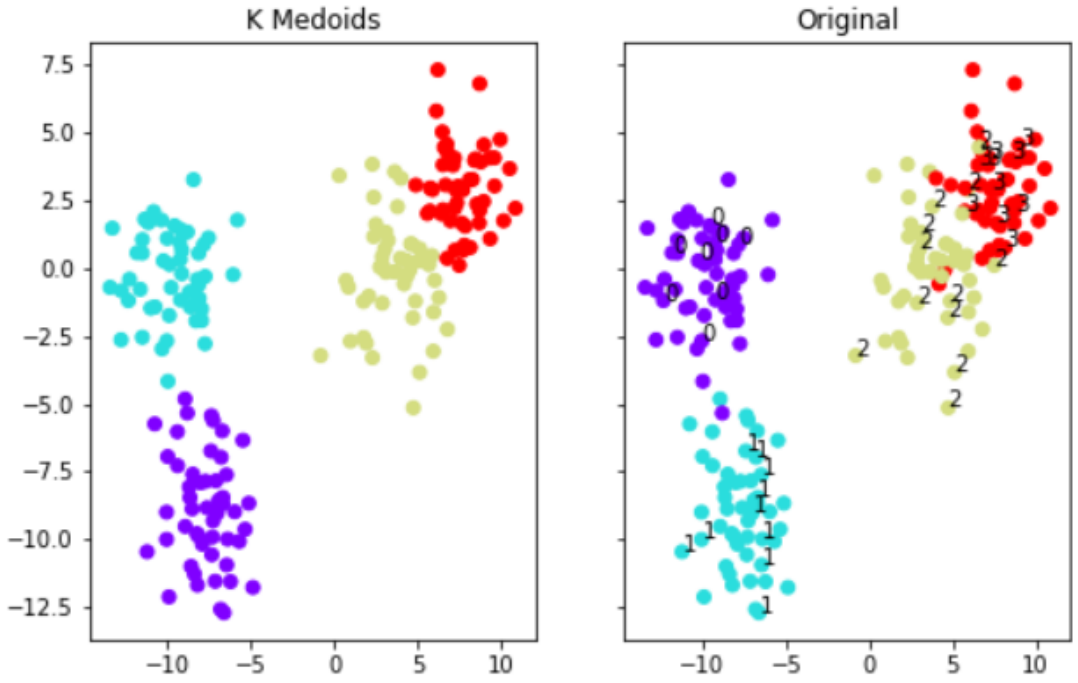
```
kmedoids.cluster_centers_
```

```
array([[-7.62795904, -8.8354951 ],
       [-9.87312876,  0.13931247],
       [ 3.65994783, -0.13260646],
       [ 7.66163195,  2.9333056 ]])
```

```
kmedoids.labels_
```

```
array([3, 0, 2, 0, 3, 1, 0, 1, 2, 2, 3, 3, 2, 3, 1, 3, 3, 3, 3, 1, 0, 2,
       3, 3, 0, 0, 2, 0, 2, 2, 3, 0, 3, 2, 3, 2, 0, 3, 0, 1, 3, 1, 3, 2,
       2, 2, 0, 0, 2, 0, 0, 2, 3, 3, 0, 2, 3, 2, 3, 1, 1, 1, 2, 2, 3, 1,
       0, 1, 1, 1, 2, 1, 2, 0, 0, 1, 1, 0, 0, 1, 3, 3, 1, 2, 0, 0, 3, 2,
       1, 0, 2, 3, 1, 2, 2, 3, 2, 1, 2, 2, 0, 3, 1, 0, 1, 2, 1, 2, 1, 1,
       3, 1, 1, 1, 2, 1, 2, 2, 0, 2, 3, 3, 3, 2, 0, 0, 3, 2, 1, 3, 0, 1,
       1, 0, 0, 0, 1, 1, 3, 0, 0, 0, 3, 2, 0, 3, 0, 1, 2, 1, 2, 2, 3, 3,
       3, 3, 3, 3, 0, 0, 0, 0, 0, 2, 3, 1, 0, 0, 3, 1, 2, 3, 3, 0, 2, 3,
       2, 0, 0, 1, 2, 0, 1, 1, 0, 1, 3, 3, 1, 2, 1, 2, 2, 0, 1, 3, 1, 0,
       1, 1], dtype=int64)
```

```
f, (ax1, ax2) = plt.subplots(1, 2, sharey=True,figsize=(8,5))
ax1.set_title('K Medoids')
ax1.scatter(X[:, 0], X[:, 1], c = kmedoids.labels_, cmap = 'rainbow')
ax2.set_title("Original")
ax2.scatter(X[:, 0], X[:, 1], c = y, cmap = 'rainbow')
for i, txt in enumerate(y):
    if i%5 == 0:
        plt.annotate(txt, (X[i, 0], X[i, 1]))
plt.savefig("KMedoids.png")
```



Unsupervised Learning - K-Medoids

Align K-Medoids prediction class with real values

```
y_pred = kmedoids.predict(X)
```

```
y_pred
```

```
array([3, 0, 2, 0, 3, 1, 0, 1, 2, 2, 3, 3, 2, 3, 1, 3, 3, 3, 3, 1, 0, 2,
       3, 3, 0, 0, 2, 0, 2, 2, 3, 0, 3, 2, 3, 2, 0, 3, 0, 1, 3, 1, 3, 2,
       2, 2, 0, 0, 2, 0, 0, 2, 3, 3, 0, 2, 3, 2, 3, 1, 1, 1, 2, 2, 3, 1,
       0, 1, 1, 1, 2, 1, 2, 0, 0, 1, 1, 0, 0, 1, 3, 3, 1, 2, 0, 0, 3, 2,
       1, 0, 2, 3, 1, 2, 2, 3, 2, 1, 2, 2, 0, 3, 1, 0, 1, 2, 1, 2, 1, 1,
       3, 1, 1, 1, 2, 1, 2, 2, 0, 2, 3, 3, 3, 2, 0, 0, 3, 2, 1, 3, 0, 1,
       1, 0, 0, 0, 1, 1, 3, 0, 0, 0, 3, 2, 0, 3, 0, 1, 2, 1, 2, 2, 3, 3,
       3, 3, 3, 3, 0, 0, 0, 0, 0, 2, 3, 1, 0, 0, 3, 1, 2, 3, 3, 0, 2, 3,
       2, 0, 0, 1, 2, 0, 1, 1, 0, 1, 3, 3, 1, 2, 1, 2, 2, 0, 1, 3, 1, 0,
       1, 1], dtype=int64)
```

```
y
```

```
array([2, 1, 2, 1, 3, 0, 1, 0, 2, 2, 3, 3, 2, 3, 0, 3, 3, 3, 3, 0, 1, 2,
       3, 3, 1, 1, 3, 1, 2, 2, 3, 1, 3, 3, 3, 2, 1, 3, 1, 0, 3, 0, 3, 2,
       2, 2, 1, 1, 2, 1, 1, 2, 3, 3, 1, 2, 3, 2, 3, 0, 0, 0, 2, 2, 3, 0,
       1, 0, 0, 0, 2, 0, 2, 1, 1, 0, 0, 1, 1, 0, 2, 3, 0, 2, 1, 1, 3, 2,
       0, 1, 2, 3, 0, 2, 2, 3, 2, 0, 2, 2, 1, 3, 0, 1, 0, 2, 0, 2, 0, 0,
       3, 0, 0, 0, 2, 0, 2, 2, 1, 2, 3, 3, 3, 2, 1, 1, 3, 2, 0, 3, 1, 0,
       0, 1, 1, 1, 0, 0, 3, 1, 1, 1, 3, 2, 1, 3, 1, 0, 2, 0, 2, 2, 3, 3,
       2, 3, 3, 3, 0, 1, 1, 1, 1, 2, 3, 0, 1, 1, 3, 0, 2, 3, 3, 1, 2, 3,
       2, 1, 1, 0, 2, 1, 0, 0, 1, 0, 3, 3, 0, 2, 0, 3, 2, 1, 0, 2, 0, 1,
       0, 0])
```

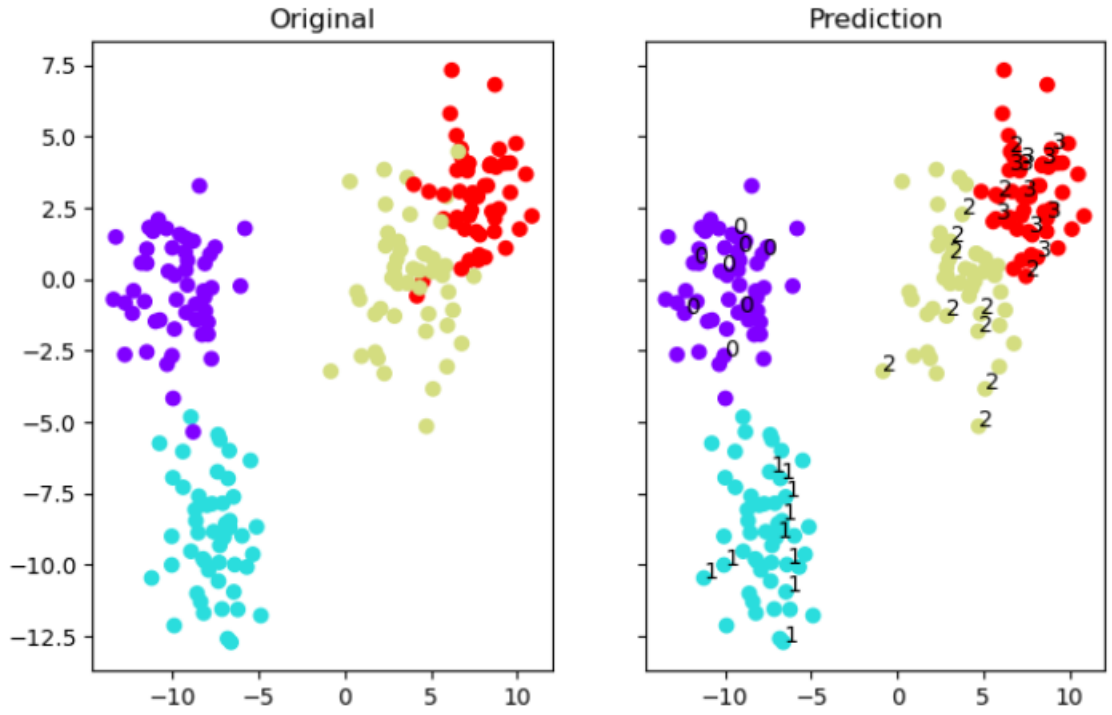
```
y_pred = np.where(y_pred==1, 10, y_pred)
y_pred = np.where(y_pred==0, 1, y_pred)
y_pred = np.where(y_pred==10, 0, y_pred)
```

```
y_pred
```

```
array([3, 1, 2, 1, 3, 0, 1, 0, 2, 2, 3, 3, 2, 3, 0, 3, 3, 3, 3, 0, 1, 2,
       3, 3, 1, 1, 2, 1, 2, 2, 3, 1, 3, 2, 3, 2, 1, 3, 1, 0, 3, 0, 3, 2,
       2, 2, 1, 1, 2, 1, 1, 2, 3, 3, 1, 2, 3, 2, 3, 0, 0, 0, 2, 2, 3, 0,
       1, 0, 0, 0, 2, 0, 2, 1, 1, 0, 0, 1, 1, 0, 3, 3, 0, 2, 1, 1, 3, 2,
       0, 1, 2, 3, 0, 2, 2, 3, 2, 0, 2, 2, 1, 3, 0, 1, 0, 2, 0, 2, 0, 0,
       3, 0, 0, 0, 2, 0, 2, 2, 1, 2, 3, 3, 3, 2, 1, 1, 3, 2, 0, 3, 1, 0,
       0, 1, 1, 1, 0, 0, 3, 1, 1, 1, 3, 2, 1, 3, 1, 0, 2, 0, 2, 2, 3, 3,
       3, 3, 3, 3, 1, 1, 1, 1, 1, 2, 3, 0, 1, 1, 3, 0, 2, 3, 3, 1, 2, 3,
       2, 1, 1, 0, 2, 1, 0, 0, 1, 0, 3, 3, 0, 2, 0, 2, 2, 1, 0, 3, 0, 1,
       0, 0], dtype=int64)
```

Unsupervised Learning - K-Medoids


```
f, (ax1, ax2) = plt.subplots(1, 2, sharey = True, figsize = (8, 5))
ax1.set_title('Original')
ax1.scatter(X[:, 0], X[:, 1], c = y, cmap = 'rainbow')
ax2.set_title("Prediction")
ax2.scatter(X[:, 0], X[:, 1], c = y_pred, cmap = 'rainbow')
for i, txt in enumerate(y):
    if i%5 == 0:
        plt.annotate(txt, (X[i, 0], X[i, 1]))
plt.savefig("KMedoid_pred.png")
```




```
print(confusion_matrix(y, y_pred))
```

```
[[49  1  0  0]
 [ 0 50  0  0]
 [ 0  0 46  4]
 [ 0  0  3 47]]
```

```
print(classification_report(y, y_pred))
```

	precision	recall	f1-score	support
0	1.00	0.98	0.99	50
1	0.98	1.00	0.99	50
2	0.94	0.92	0.93	50
3	0.92	0.94	0.93	50
accuracy			0.96	200
macro avg	0.96	0.96	0.96	200
weighted avg	0.96	0.96	0.96	200

Unsupervised Learning - K-Means vs. K-Medoids

K-Means

```
print(confusion_matrix(y, y_pred))
```

```
[[49  1  0  0]
 [ 0 50  0  0]
 [ 0  0 46  4]
 [ 0  0  3 47]]
```

```
print(classification_report(y, y_pred))
```

	precision	recall	f1-score	support
0	1.00	0.98	0.99	50
1	0.98	1.00	0.99	50
2	0.94	0.92	0.93	50
3	0.92	0.94	0.93	50
accuracy			0.96	200
macro avg	0.96	0.96	0.96	200
weighted avg	0.96	0.96	0.96	200

K-Medoids

```
print(confusion_matrix(y, y_pred))
```

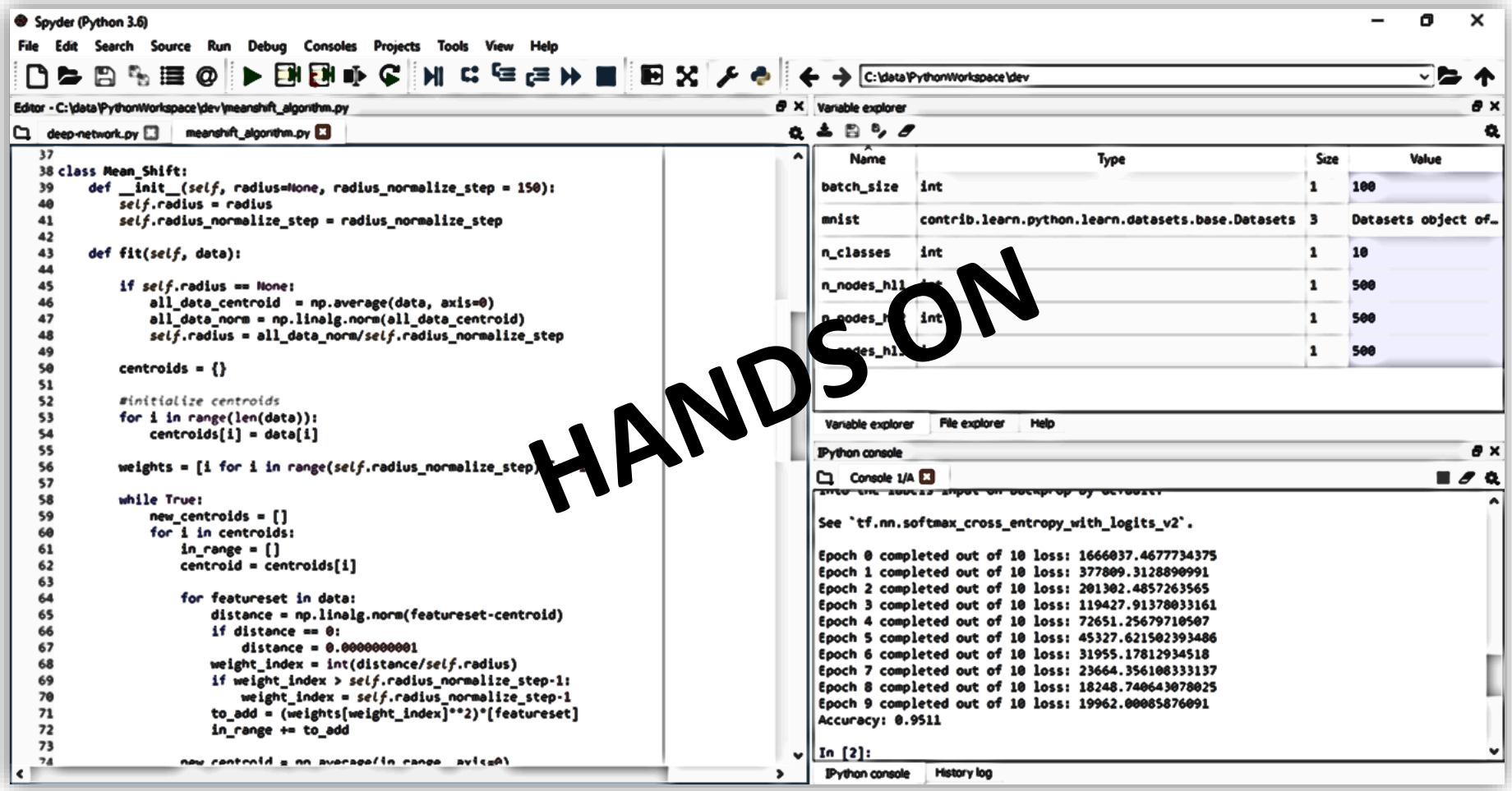
```
[[49  1  0  0]
 [ 0 50  0  0]
 [ 0  0 46  4]
 [ 0  0  3 47]]
```

```
print(classification_report(y, y_pred))
```

	precision	recall	f1-score	support
0	1.00	0.98	0.99	50
1	0.98	1.00	0.99	50
2	0.94	0.92	0.93	50
3	0.92	0.94	0.93	50
accuracy			0.96	200
macro avg	0.96	0.96	0.96	200
weighted avg	0.96	0.96	0.96	200

Can we compare these two models?

Hands On


Spyder (Python 3.6)
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\data\PythonWorkspace\dev
Editor - C:\data\PythonWorkspace\dev\meanshift_algorithm.py
deep-network.py
meanshift_algorithm.py
class Mean_Shift:
def __init__(self, radius=None, radius_normalize_step = 150):
self.radius = radius
self.radius_normalize_step = radius_normalize_step
def fit(self, data):
if self.radius == None:
all_data_centroid = np.average(data, axis=0)
all_data_norm = np.linalg.norm(all_data_centroid)
self.radius = all_data_norm/self.radius_normalize_step
centroids = {}
#initialize centroids
for i in range(len(data)):
centroids[i] = data[i]
weights = [i for i in range(self.radius_normalize_step)]
while True:
new_centroids = []
for i in centroids:
in_range = []
centroid = centroids[i]
for featureset in data:
distance = np.linalg.norm(featureset-centroid)
if distance == 0:
distance = 0.0000000001
weight_index = int(distance/self.radius)
if weight_index > self.radius_normalize_step-1:
weight_index = self.radius_normalize_step-1
to_add = (weights[weight_index]**2)*[featureset]
in_range += to_add
Variable explorer
Name Type Size Value
batch_size int 1 100
mnist contrib.learn.python.learn.datasets.base.Datasets 3 Datasets object of…
n_classes int 1 10
1 500
1 500
1 500
Variable explorer File explorer Help
IPython console
Console 1/A
See `tf.nn.softmax_cross_entropy_with_logits_v2`.
Epoch 0 completed out of 10 loss: 1666037.4677734375
Epoch 1 completed out of 10 loss: 377809.3128890991
Epoch 2 completed out of 10 loss: 201302.4857263565
Epoch 3 completed out of 10 loss: 119427.91378033161
Epoch 4 completed out of 10 loss: 72651.25679710507
Epoch 5 completed out of 10 loss: 45327.621502393486
Epoch 6 completed out of 10 loss: 31955.17812934518
Epoch 7 completed out of 10 loss: 23664.356108333137
Epoch 8 completed out of 10 loss: 18248.740643078025
Epoch 9 completed out of 10 loss: 19962.00085876091
Accuracy: 0.9511
In [2]:
IPython console History log
HANDS ON