HO-4

November 30, 2018

1 HO-4 Report

1.0.1 Irving Hernández Gallegos

1.0.2 Objective

Discuss different techniques for unsupervised learning and will focus on several clustering techniques.

```
- Consider basic concepts like distance and similarity, taxonomy of clustering techniques and go
- Explore three basic clustering techniques, namely, K-means, spectral clustering and hierarchic
- Illustrate the use of clustering techniques on a real problem: defining groups of countries ac
```

http://vargas-solar.com/data-centric-smart-everything/hands-on/unsupervised-learning-comparing-clustering-methods/

1.0.3 Clustering

Partition unlabeled examples into disjoint subsets of clusters, such that:

```
Examples within a cluster are similar (high intra-class similarity).
Examples in different clusters are different (low inter-class similarity).
It can help in discovering new categories in an unsupervised manner (no sample category labels p
```

Similarity and distance The notion of similarity is a tough one, however we can use the notion of distance as a surrogate. The most well-known instantiations of this metric are:

```
Euclidean distance.
Manhattan distance.
Max-distance.
```

Rand index, R A clustering result satisfies homogeneity if all of its clusters contain only data points which are members of the same original (a single) class. A clustering result satisfies completeness if all the data points that are members of a given class are elements of the same automatic cluster. Both scores have positive values between 0.0 and 1.0, larger values being desirable.

```
In [1]: import matplotlib.pylab as plt
```

```
In [2]: from sklearn import metrics
        metrics.homogeneity_score([0, 0, 1, 1], [1, 1, 0, 0])

        print("%.3f" % metrics.homogeneity_score([0, 0, 1, 1], [0, 0, 1, 2]))
        print("%.3f" % metrics.homogeneity_score([0, 0, 1, 1], [0, 1, 2, 3]))
        print("%.3f" % metrics.homogeneity_score([0, 0, 1, 1], [0, 1, 0, 1]))
        print("%.3f" % metrics.homogeneity_score([0, 0, 1, 1], [0, 0, 0, 0]))
        print(metrics.completeness_score([0, 0, 1, 1], [1, 1, 0, 0]))
        print(metrics.completeness_score([0, 0, 1, 1], [0, 0, 0, 0]))
        print(metrics.completeness_score([0, 1, 2, 3], [0, 0, 1, 1]))
        print(metrics.completeness_score([0, 0, 1, 1], [0, 1, 0, 1]))
        print(metrics.completeness_score([0, 0, 0, 0], [0, 1, 2, 3]))
```

```
1.000
1.000
0.000
0.000
1.0
1.0
0.9999999999999999
0.0
0.0
```

V-measure is the harmonic mean between homogeneity and completeness: v = 2 * (homogeneity * completeness) / (homogeneity + completeness)

```
In [3]: print (metrics.v_measure_score([0, 0, 1, 1], [0, 0, 1, 1]))
        print (metrics.v_measure_score([0, 0, 1, 1], [1, 1, 0, 0]))
```

```
1.0
1.0
```

Q1: Labelings that assign all classes members to the same clusters are: *Completness*, but not *Homogeneity*:

```
In [4]: print("%.3f" % metrics.completeness_score([0, 1, 2, 3], [0, 0, 0, 0]))
        print("%.3f" % metrics.homogeneity_score([0, 1, 2, 3], [0, 0, 0, 0]))
        print("%.3f" % metrics.v_measure_score([0, 1, 2, 3], [0, 0, 0, 0]))
        print("%.3f" % metrics.v_measure_score([0, 0, 1, 2], [0, 0, 1, 1]))
        print("%.3f" % metrics.v_measure_score([0, 1, 2, 3], [0, 0, 1, 1]))
```

```
1.000
0.000
0.000
0.800
0.667
```

Q2: Labelings that have pure clusters with members coming from the same classes are *homogeneous* but un-necessary splits harms *completeness* and thus penalise V-measure as well:

```
In [5]: print("%.3f" % metrics.v_measure_score([0, 0, 1, 1], [0, 0, 1, 2]))
        print("%.3f" % metrics.v_measure_score([0, 0, 1, 1], [0, 1, 2, 3]))
```

```
0.800
0.667
```

```
In [6]: print("%.3f" % metrics.v_measure_score([0, 0, 0, 0], [0, 1, 2, 3]))
```

```
0.000
```

Q4. Clusters that include samples from totally different classes totally destroy the *homogeneity* of the labelling, hence:

```
In [7]: print("%.3f" % metrics.v_measure_score([0, 0, 1, 1], [0, 0, 0, 0]))
```

```
0.000
```

Clustering techniques: how to group samples? There are two big families of clustering techniques:

- Partitional algorithms: Start with a random partition and refine it iteratively.
- Hierarchical algorithms: Agglomerative (bottom-up), top-down.
- Partitional algorithms. They can be divided in two branches:
  - Hard partition algorithms, such as K-means, assign a unique cluster value to each element in the feature space.
  - Soft partition algorithms, such as Mixture of Gaussians, can be viewed as density estimators and assign a confidence or probability to each point in the space.

K-means algorithm Algorithm:

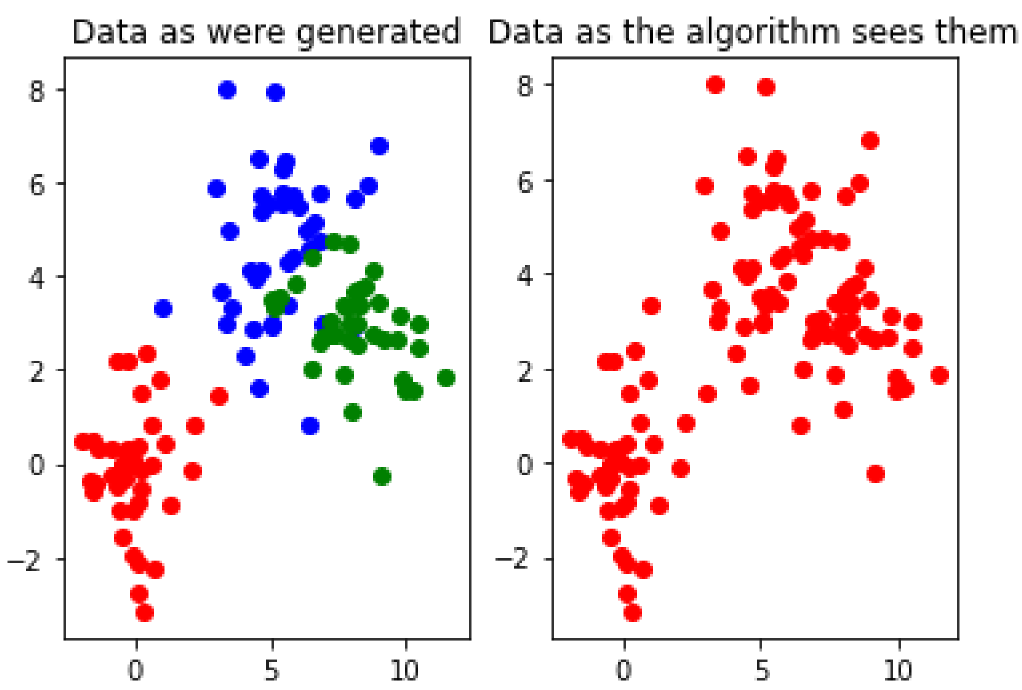
- Initialise the value K of desirable clusters.
- Initialise the K cluster centres, e.g. randomly.
- Decide the class memberships of the N data samples by assigning them to the
- nearest cluster centroids (e.g. the center of gravity or mean).
- Re-estimate the K cluster centres, by assuming the memberships found above are correct.
- If none of the N objects changed membership in the last iteration, exit. Otherwise go to 3.

```
In [8]: import numpy as np

        #Create some data
        MAXN=40
        X = np.concatenate([1.25*np.random.randn(MAXN,2), 5+1.5*np.random.randn(MAXN,2)])
        X = np.concatenate([X,[8,3]+1.2*np.random.randn(MAXN,2)])
        X.shape
```

Out[8]: (120, 2)

In [9]:
```
#Just for visualization purposes, create the labels of the 3 distributions
y = np.concatenate([np.ones((MAXN,1)),2*np.ones((MAXN,1))])
y = np.concatenate([y,3*np.ones((MAXN,1))])

plt.subplot(1,2,1)
plt.scatter(X[(y==1).ravel(),0],X[(y==1).ravel(),1],color='r')
plt.scatter(X[(y==2).ravel(),0],X[(y==2).ravel(),1],color='b')
plt.scatter(X[(y==3).ravel(),0],X[(y==3).ravel(),1],color='g')
plt.title('Data as were generated')

plt.subplot(1,2,2)
plt.scatter(X[:,0],X[:,1],color='r')
plt.title('Data as the algorithm sees them')

plt.savefig("files/ch07/sample.png",dpi=300, bbox_inches='tight')

from sklearn import cluster

K=3 # Assuming to be 3 clusters!

clf = cluster.KMeans(init='random', n_clusters=K)
clf.fit(X)
```

Out[9]:
```
KMeans(algorithm='auto', copy_x=True, init='random', max_iter=300,
    n_clusters=3, n_init=10, n_jobs=1, precompute_distances='auto',
    random_state=None, tol=0.0001, verbose=0)
```



Each clustering algorithm comes in two variants: - a class, that implements the fit method to learn the clusters on train data, - a predict function, that, given test data, returns an array of integer labels corresponding to the different clusters.

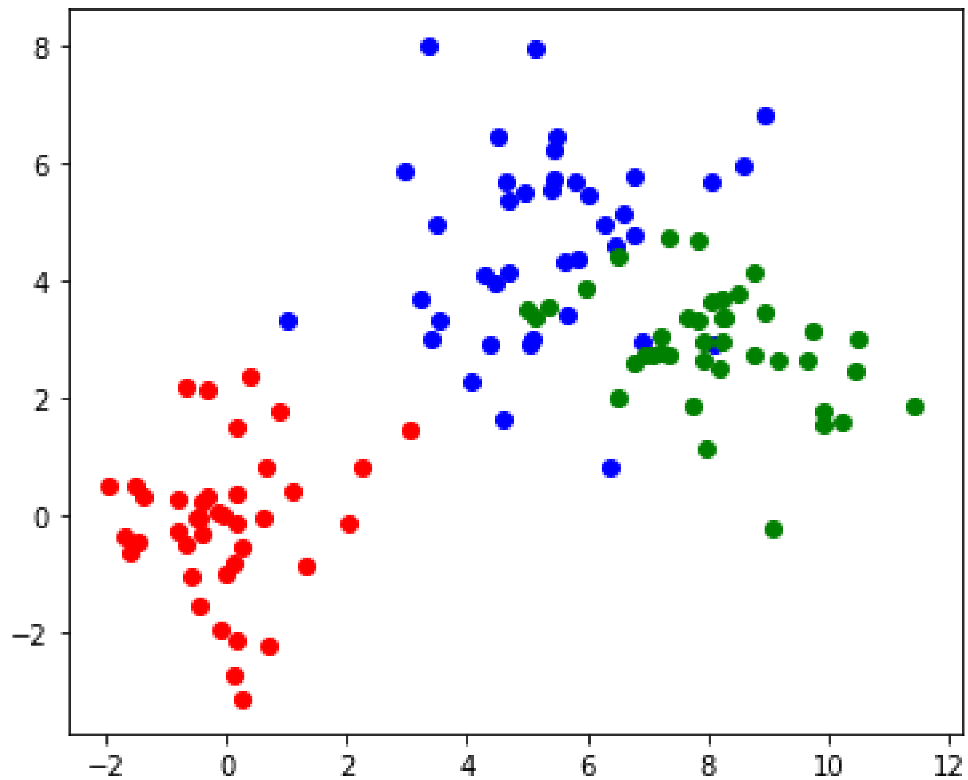
```
In [10]: print (clf.labels_) # or
         print (clf.predict(X)) # equivalent
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 2 2 1 2 2 2 2 0 2 2 2 2 2 2 1 2 2 1 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2
 1 1 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 2 1 1 2 1 1 1 1 1 2 1
 1 1 1 1 2 1 1 1 1]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 2 2 1 2 2 2 2 0 2 2 2 2 2 2 1 2 2 1 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2
 1 1 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 2 1 1 2 1 1 1 1 1 2 1
 1 1 1 1 2 1 1 1 1]
```

```
In [11]: print (X[(y==1).ravel(),0]) #numpy.ravel() returns a flattened array
         print (X[(y==1).ravel(),1])
```

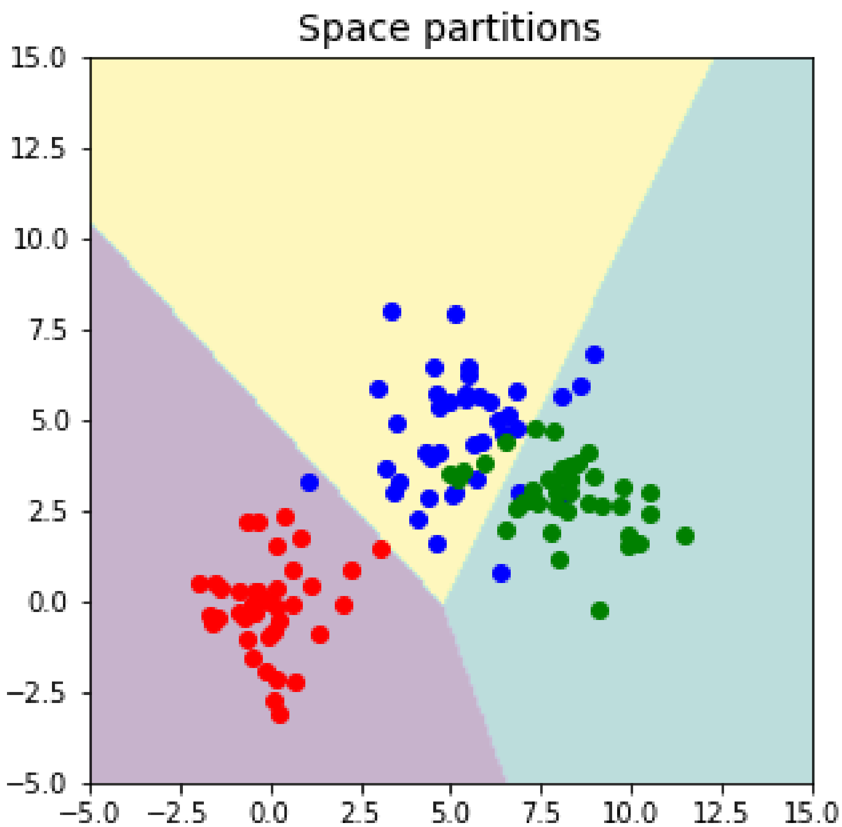
```
[-0.40545708  0.15394283  0.15786027 -0.82062541 -0.41557599  0.63581042
 -1.58675335 -0.44665755 -0.08144315 -0.0725023  -1.44816273 -0.82697549
 -0.66898602  0.13316902  0.86303318 -0.02701064 -0.45143153 -0.59559329
 -1.69101836 -1.94082241  0.18505239  3.05745451 -1.50985393  0.2579971
 -0.30540207  0.27797648  0.17954433  0.61251416  1.10542395 -0.31554709
  2.04451193 -0.66011005  0.10818125 -0.11960933  0.40382846  0.68390498
  1.33327742 -0.49317339  2.23138862 -1.36070912]
[ 0.24963028  0.38440249 -2.13664873  0.29919732 -0.30905969  0.8378455
 -0.61379286 -0.05600474 -1.95564476 -0.00582765 -0.44291491 -0.275059
 -0.48502197 -0.81919979  1.77540112 -0.97575205 -1.55825904 -1.02238161
 -0.35309647  0.49419134  1.50425277  1.4505941   0.5074627  -0.53190613
  0.30676209 -3.13532449 -0.12938985 -0.04852249  0.41752181  2.16215447
 -0.11386552  2.17363418 -2.73510813  0.06814037  2.37219378 -2.23742441
 -0.87181978 -0.0527498   0.84857257  0.32768029]
```

```
In [12]: plt.scatter(X[(y==1).ravel(),0],X[(y==1).ravel(),1],color='r')
         plt.scatter(X[(y==2).ravel(),0],X[(y==2).ravel(),1],color='b')
         plt.scatter(X[(y==3).ravel(),0],X[(y==3).ravel(),1],color='g')

         fig = plt.gcf()
         fig.set_size_inches((6,5))
```




```
In [13]: x = np.linspace(-5,15,200)
         XX,YY = np.meshgrid(x,x)
         sz=XX.shape
         data=np.c_[XX.ravel(),YY.ravel()]
         # c_ translates slice objects to concatenation along the second axis.

In [14]: Z=clf.predict(data) # returns the labels of the data
         print (Z)

[0 0 0 ... 1 1 1]


In [15]: # Visualize space partition
         plt.imshow(Z.reshape(sz), interpolation='bilinear', origin='lower',
         extent=(-5,15,-5,15),alpha=0.3, vmin=0, vmax=K-1)
         plt.title('Space partitions', size=14)
         plt.scatter(X[(y==1).ravel(),0],X[(y==1).ravel(),1],color='r')
         plt.scatter(X[(y==2).ravel(),0],X[(y==2).ravel(),1],color='b')
         plt.scatter(X[(y==3).ravel(),0],X[(y==3).ravel(),1],color='g')

         fig = plt.gcf()
         fig.set_size_inches((6,5))
```

```
plt.savefig("files/ch07/samples3.png",dpi=300, bbox_inches='tight')
```

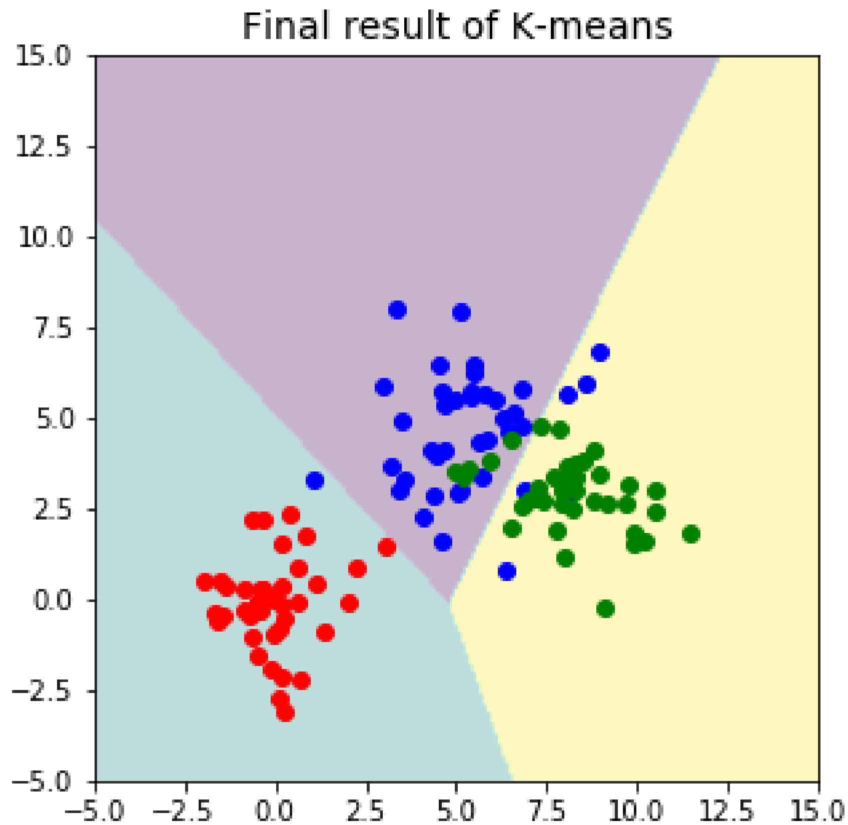



```
In [16]: clf = cluster.KMeans(n_clusters=K, random_state=0)
         #initialize the k-means clustering
         clf.fit(X) #run the k-means clustering

         data=np.c_[XX.ravel(),YY.ravel()]
         Z=clf.predict(data) # returns the clustering labels of the data
```

Visualising true labels by coloured points and space tessellation:

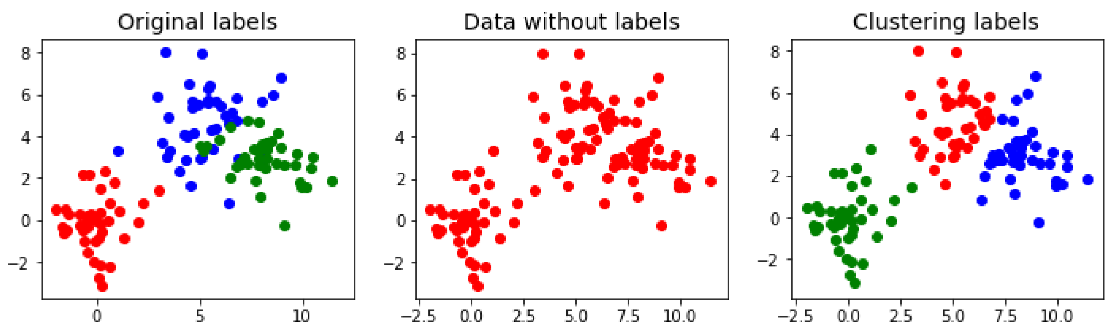
```
In [17]: plt.title('Final result of K-means', size=14)

         plt.scatter(X[(y==1).ravel(),0],X[(y==1).ravel(),1],color='r')
         plt.scatter(X[(y==2).ravel(),0],X[(y==2).ravel(),1],color='b')
         plt.scatter(X[(y==3).ravel(),0],X[(y==3).ravel(),1],color='g')

         plt.imshow(Z.reshape(sz), interpolation='bilinear', origin='lower',
         extent=(-5,15,-5,15),alpha=0.3, vmin=0, vmax=K-1)
```

```
x = np.linspace(-5,15,200)
XX,YY = np.meshgrid(x,x)
fig = plt.gcf()
fig.set_size_inches((6,5))

plt.savefig("files/ch07/randscore.png",dpi=300, bbox_inches='tight')
```



In [18]:
```
clf = cluster.KMeans(init='random', n_clusters=K, random_state=0)
#initialize the k-means clustering
clf.fit(X) #run the k-means clustering
Zx=clf.predict(X)

plt.subplot(1,3,1)
plt.title('Original labels', size=14)
plt.scatter(X[(y==1).ravel(),0],X[(y==1).ravel(),1],color='r')
plt.scatter(X[(y==2).ravel(),0],X[(y==2).ravel(),1],color='b') # b
plt.scatter(X[(y==3).ravel(),0],X[(y==3).ravel(),1],color='g') # g
fig = plt.gcf()
fig.set_size_inches((12,3))

plt.subplot(1,3,2)
```

```
plt.title('Data without labels', size=14)
plt.scatter(X[(y==1).ravel(),0],X[(y==1).ravel(),1],color='r')
plt.scatter(X[(y==2).ravel(),0],X[(y==2).ravel(),1],color='r') # b
plt.scatter(X[(y==3).ravel(),0],X[(y==3).ravel(),1],color='r') # g
fig = plt.gcf()
fig.set_size_inches((12,3))

plt.subplot(1,3,3)
plt.title('Clustering labels', size=14)
plt.scatter(X[(Zx==1).ravel(),0],X[(Zx==1).ravel(),1],color='r')
plt.scatter(X[(Zx==2).ravel(),0],X[(Zx==2).ravel(),1],color='b')
plt.scatter(X[(Zx==0).ravel(),0],X[(Zx==0).ravel(),1],color='g')
fig = plt.gcf()
fig.set_size_inches((12,3))
```



Q.4: Shall the centroids belong to the original set of points? The K-means algorithm aims to choose centroids minimising a criterion known as the inertia or within-cluster sum-of-squares:

Summary

```
(+) Select good seeds using a heuristic (e.g. seeds with large distance among them).
(+) Try out multiple starting points.
(+) Initialize with the results of another method.
(-) Tends to look for spherical clusters.
(-) Prone to local minima stabilization.
```

```
In [19]: from sklearn import metrics

         clf = cluster.KMeans(n_clusters=K, init='k-means++', random_state=0,
         max_iter=300, n_init=10)
         #initialize the k-means clustering
         clf.fit(X) #run the k-means clustering

         print ('Final evaluation of the clustering:')
```

```python
print('Inertia: %.2f' % clf.inertia_)

print('Adjusted_rand_score %.2f' % metrics.adjusted_rand_score(y.ravel(),
clf.labels_))

print('Homogeneity %.2f' % metrics.homogeneity_score(y.ravel(),
clf.labels_))

print('Completeness %.2f' % metrics.completeness_score(y.ravel(),
clf.labels_))

print('V_measure %.2f' % metrics.v_measure_score(y.ravel(), clf.labels_))

print('Silhouette %.2f' % metrics.silhouette_score(X, clf.labels_,
metric='euclidean'))

clf1 = cluster.KMeans(n_clusters=K, init='random', random_state=0,
max_iter=2, n_init=2)
#initialize the k-means clustering
clf1.fit(X) #run the k-means clustering

print ('Final evaluation of the clustering:')

print ('Inertia: %.2f' % clf1.inertia_)

print ('Adjusted_rand_score %.2f' % metrics.adjusted_rand_score(y.ravel(),
clf1.labels_))

print ('Homogeneity %.2f' % metrics.homogeneity_score(y.ravel(),
clf1.labels_))

print ('Completeness %.2f' % metrics.completeness_score(y.ravel(),
clf1.labels_))

print ('V_measure %.2f' % metrics.v_measure_score(y.ravel(),
clf1.labels_))

print ('Silhouette %.2f' % metrics.silhouette_score(X, clf1.labels_,
metric='euclidean'))
```

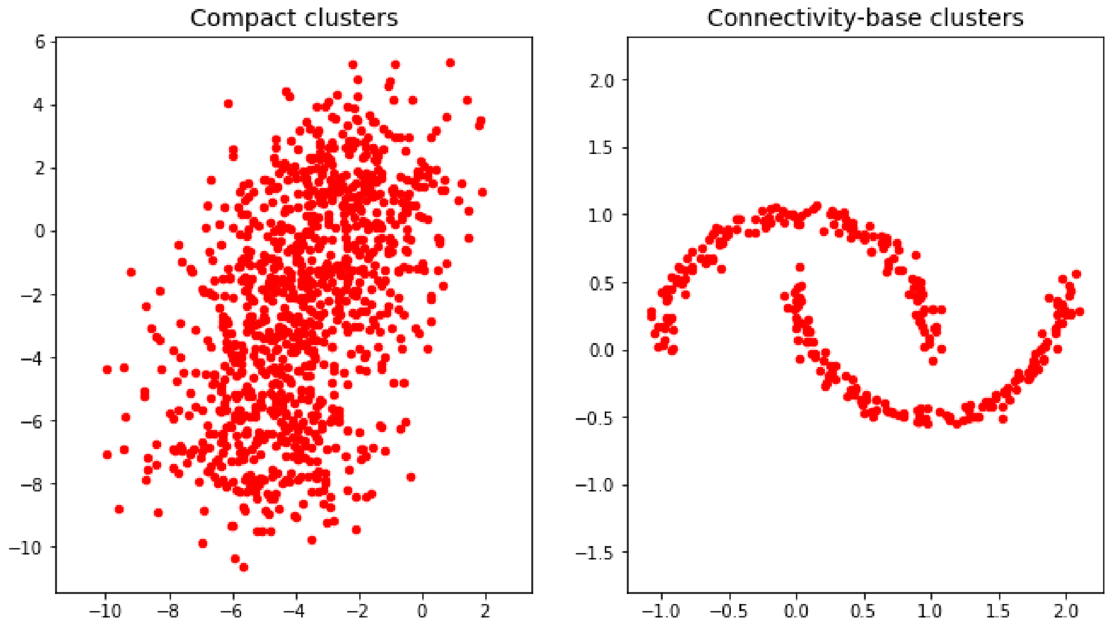
```
Final evaluation of the clustering:
Inertia: 359.40
Adjusted_rand_score 0.74
Homogeneity 0.72
Completeness 0.72
V_measure 0.72
Silhouette 0.53
Final evaluation of the clustering:
```

```
Inertia: 408.88
Adjusted_rand_score 0.70
Homogeneity 0.70
Completeness 0.70
V_measure 0.70
Silhouette 0.52
```

Spectral clustering Spectral clustering refers to a family of methods that use spectral techniques. Specifically, these techniques are related to the eigen-decomposition of an affinity or similarity matrix and attempt to solve the problem of clustering according to connectivity. Let us consider an ideal similarity matrix of two clear sets.

Let us illustrate it on some examples with non Gaussian distribution. Scikit-learn has a library to generate datasets with different shapes like moons, blobs, etc.

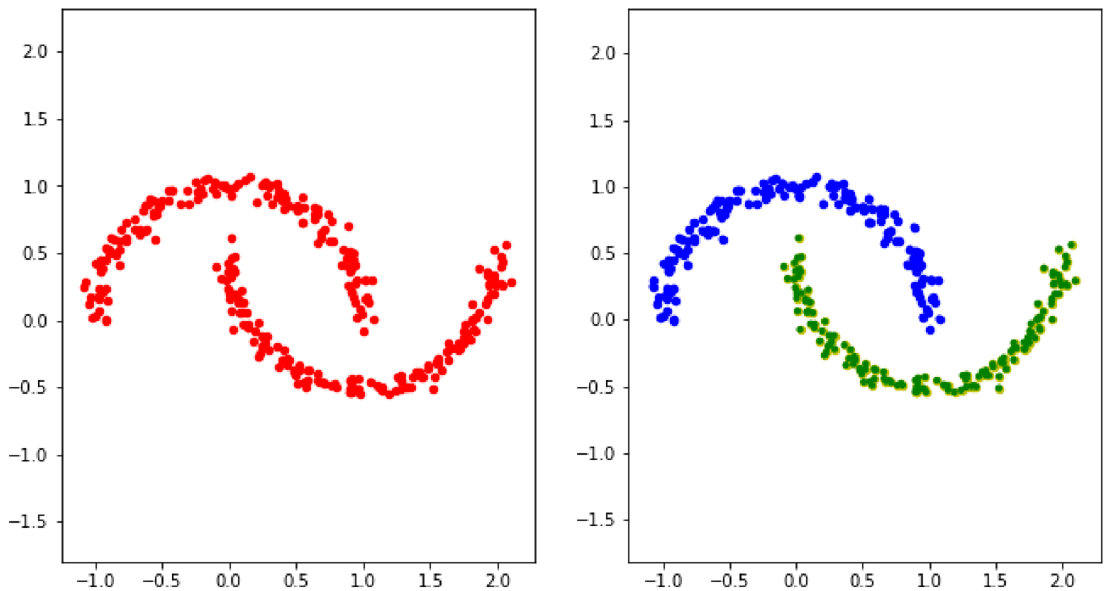
```
In [20]: from sklearn.datasets.samples_generator import make_blobs,make_moons

         X, labels_true = make_blobs(n_samples=1000, centers=3, cluster_std=[1.7,1.7,1.7])

         plt.subplot(1,2,1)
         plt.scatter(X[:, 0], X[:, 1], c='r', marker='o',s=20)
         plt.axis('equal')
         plt.title('Compact clusters',size=14)

         [Xmoons, ymoons] = make_moons(n_samples=300, noise=.05)
         plt.subplot(1,2,2)
         plt.scatter(Xmoons[:, 0], Xmoons[:, 1], c='r', marker='o',s=20)
         plt.axis('equal')
         plt.title('Connectivity-base clusters', size=14)
         fig = plt.gcf()
         fig.set_size_inches((11,6))

         #Let us apply the Spectral clustering to the two moons toy problem.
```




```
In [31]: from sklearn.preprocessing import StandardScaler
         from sklearn.neighbors import kneighbors_graph
         from sklearn.metrics import euclidean_distances

         colors = np.array([x for x in 'bgrcmykbgrcmykbgrcmykbgrcmyk'])
         colors = np.hstack([colors] * 20)

         # normalize dataset for easier parameter selection
         X = StandardScaler().fit_transform(Xmoons)

         # Compute distances
         distances = euclidean_distances(Xmoons)

         spectral = cluster.SpectralClustering(n_clusters=2,
         affinity="nearest_neighbors")

         spectral.fit(Xmoons)
         y_pred = spectral.labels_.astype(np.int)

In [33]: plt.subplot(1,2,1)
         plt.scatter(Xmoons[:, 0], Xmoons[:, 1], c='r', marker='o', s=20)
         plt.axis('equal')

         plt.subplot(1,2,2)
         plt.scatter(Xmoons[y_pred==0, 0], Xmoons[y_pred==0, 1], c='b',marker='o',s=20)
         plt.scatter(Xmoons[y_pred==1, 0], Xmoons[y_pred==1, 1], c='y',marker='o',s=20)
         plt.axis('equal')
```

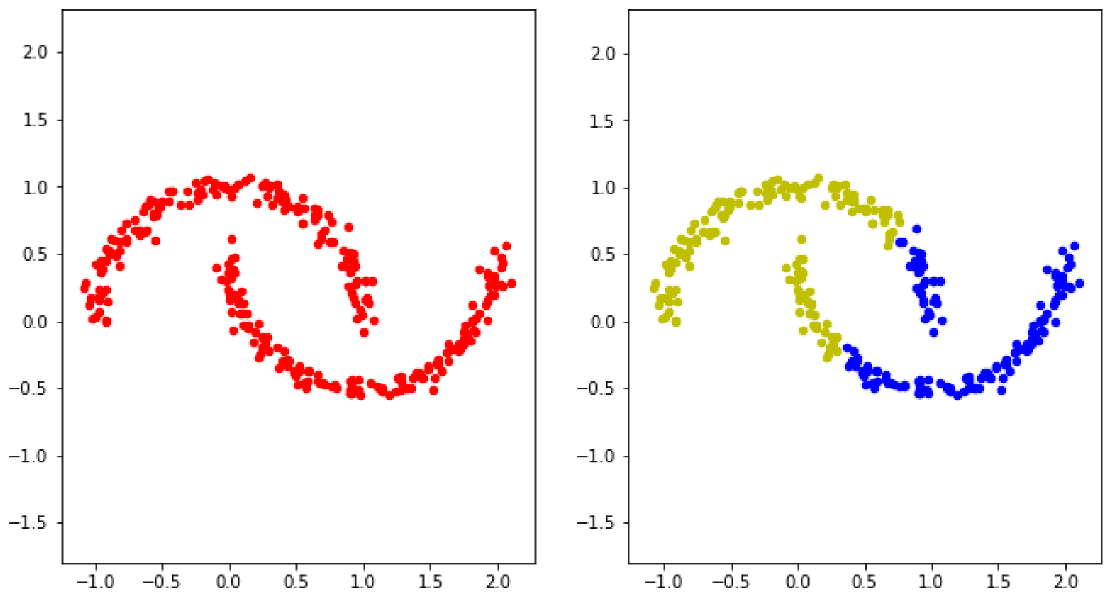
```
fig=plt.gcf()
fig.set_size_inches((11,6))
plt.scatter(Xmoons[:, 0], Xmoons[:, 1], color=colors[y_pred].tolist(),s=10)
```

Out[33]: <matplotlib.collections.PathCollection at 0x7f8f626f2048>



Let us compare the result to the K-means:

In [34]:
```
plt.subplot(1,2,1)
plt.scatter(Xmoons[:, 0], Xmoons[:, 1], c='r', marker='o',s=20)
plt.axis('equal')

# Cluster using k-means
clf = cluster.KMeans(n_clusters=2,init='k-means++')
clf.fit(Xmoons)
y_pred=clf.predict(Xmoons)

# Visualize k-means result
plt.subplot(1,2,2)
plt.scatter(Xmoons[y_pred==0, 0], Xmoons[y_pred==0, 1], c='b', marker='o',s=20)
plt.scatter(Xmoons[y_pred==1, 0], Xmoons[y_pred==1, 1], c='y', marker='o',s=20)
plt.axis('equal')
fig=plt.gcf()
fig.set_size_inches((11,6))
```



Note that since K-means looks for spheric clusters, it is unable to separate the two moon data, in contrast to the spectral clustering. ##### Hierarchical clustering Hierarchical clustering is a general family of clustering algorithms that build nested clusters by merging or splitting them successively. This hierarchy of clusters is represented as a tree (or dendrogram). The root of the tree is the unique cluster that gathers all the samples, the leaves being the clusters with only one sample. This is a nice tool, because of its interpretability. The result of the technique is a tree showing the *similarity among the samples*.

Bottom-Up agglomerative clustering sketch of algorithm

1. Starts with each sample data in a separate cluster.
2. Then, repeatedly joins the closest pair of clusters.
3. Until there is only one cluster.

The history of merging forms a binary tree or hierarchy.

Top-Down divisive clustering sketch of algorithm

1. Starting with all the data in a single cluster.
2. Consider every possible way to divide the cluster into two. Choose the best division.
3. Recursively operate on both sides.

The **Agglomerative Clustering** performs a hierarchical clustering using a bottom up approach: each observation starts in its own cluster, and clusters are successively merged together.

Defining the similarity of two clusters:

The linkage criterion determines the metric used for the merge strategy:

- Maximum or complete linkage minimises the maximum distance between observations of pairs of clusters. Based on the similarity of the two least similar members, it will give tight spherical clusters.

- Average linkage averages similarity between members i.e. minimises the average of the distances between all observations of pairs of clusters.
- Single linkage works on the similarity of two most similar members. It can create chain effects, such as follow the nearest neighbour.
- Ward minimizes the sum of squared differences within all clusters. It is a variance-minimizing approach and in this sense is similar to the k-means objective function but tackled with an agglomerative hierarchical approach.

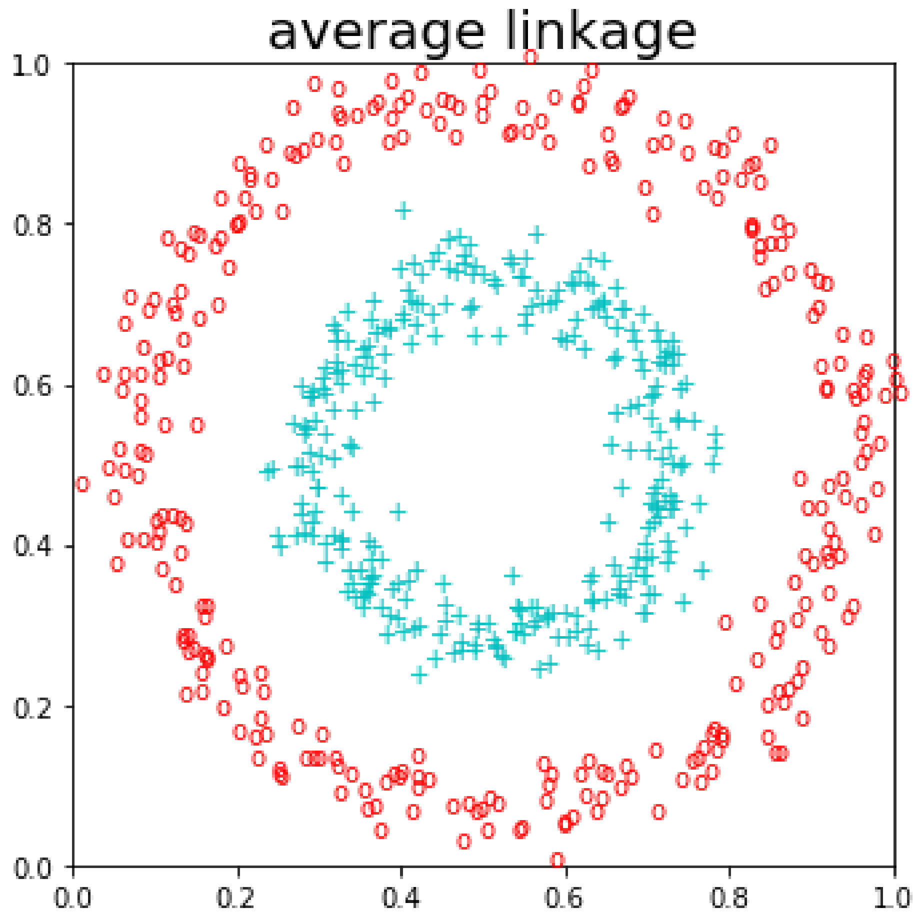
Agglomerative Clustering can also scale to large number of samples when it is used jointly with a connectivity matrix, but is computationally expensive when no connectivity constraints are added between samples: it considers at each step all the possible merges.

Let us illustrate how the different linkages work with an example. Let us generate three clusters as follows:

```
In [35]: MAXN1 =500
         MAXN2 =400
         MAXN3 =300
         X1 = np.concatenate ([2.25*np.random.randn(MAXN1,2),4+1.7* np.random.randn (MAXN2 ,2)])
         X1 = np.concatenate ([X1,[8,3]+1.9* np.random.randn(MAXN3 ,2)])
         y1 = np.concatenate ([ np.ones ((MAXN1,1)),2* np.ones((MAXN2,1))])
         y1 = np.concatenate ([y1,3* np.ones((MAXN3,1))]).ravel()
         y1 = np.int_(y1)
         labels_y1=['+','*','o']
         colors=['r','g', 'b']
```

Let us apply agglomerative clustering using the different linkages:

```
In [58]: import time
         from sklearn.cluster import AgglomerativeClustering

         for linkage in ('ward', 'complete', 'average'):
             clustering = AgglomerativeClustering(linkage=linkage,n_clusters =3)
             clustering.fit(X1)

         x_min , x_max = np.min(X1, axis =0) , np.max(X1,axis =0)
         X1 = (X1 - x_min ) / ( x_max - x_min )
         fig = plt.figure ()
         fig.set_size_inches((5,5))

         for i in range (X1.shape [0]) :
             plt.text(X1[i,0],X1[i,1],labels_y1[y1[i]-1],color=colors[y1[i]-1])

         plt.title ("%s linkage" % linkage,size =20)
         plt.tight_layout()
         plt.savefig("files/ch07/%slinkage.png" % linkage,dpi=300, bbox_inches='tight')

         plt.show()
```

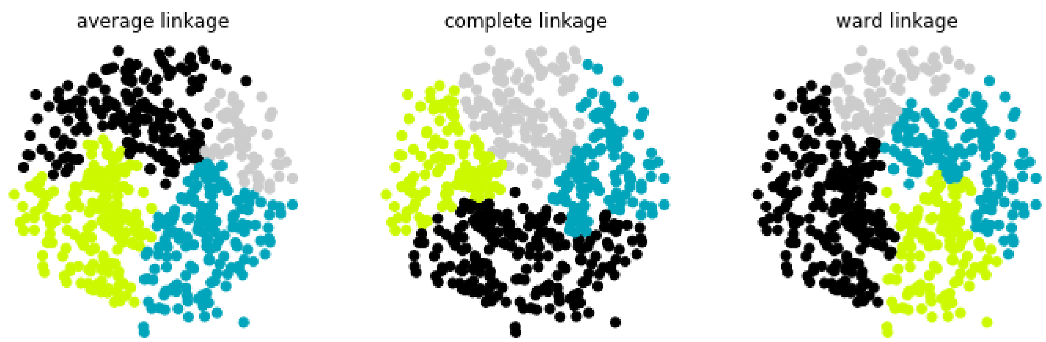


Agglomerative clustering has a "rich get richer" behaviour that leads to uneven cluster sizes. In this regard, complete linkage is the worst strategy, and Ward gives the most regular sizes. However, the affinity cannot be varied with Ward, thus for non Euclidean metrics, average linkage is a good alternative. Let us illustrate the performance on some other datasets with more complex distributions:

```
In [38]: from sklearn import cluster, datasets
         from sklearn.cluster import AgglomerativeClustering

         [X1, y1] = datasets.make_circles(n_samples=600, factor=.5, noise=.05)
         [X2, y2] = datasets.make_circles(n_samples=600, factor=.5, noise=.15)

         n_clusters=4

         for X in [X1,X2]:
             plt.figure(figsize=(12, 4))

         for index, linkage in enumerate(('average', 'complete', 'ward')):
             plt.subplot(1, 3, index + 1)
```

```
        model = AgglomerativeClustering(linkage=linkage,n_clusters=n_clusters)
        model.fit(X)
        plt.scatter(X[:, 0], X[:, 1], c=model.labels_, cmap=plt.cm.nipy_spectral)
        plt.title('%s linkage' % linkage,fontdict=dict(verticalalignment='top'))
        plt.axis('equal')
        plt.axis('off')

    plt.show()
```

```
<Figure size 864x288 with 0 Axes>
```

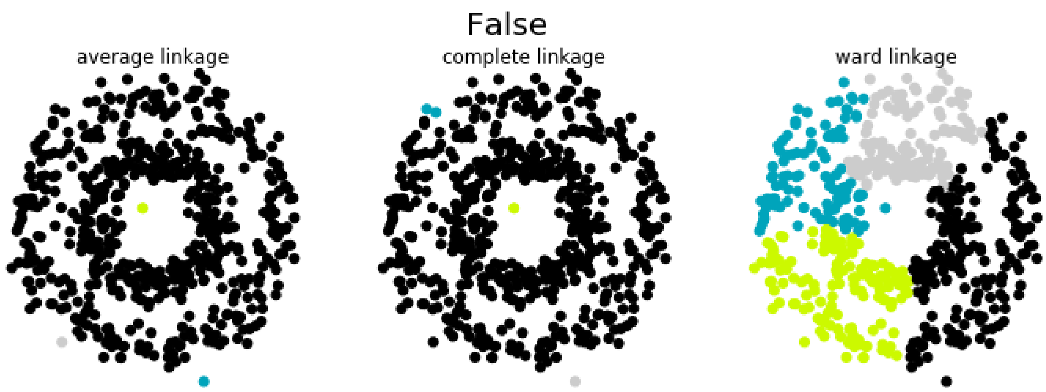


Adding connectivity constraints

```
In [42]: [X1, y1] = datasets.make_circles(n_samples=600, factor=.5, noise=.07)
         [X2, y2] = datasets.make_circles(n_samples=600, factor=.5, noise=.1)

         for X in [X1,X2]:
             knn_graph = kneighbors_graph(X, 5)
         for connectivity in (None, knn_graph):
             plt.figure(figsize=(12, 4))
         for index, linkage in enumerate(('average', 'complete', 'ward')):
             plt.subplot(1, 3, index + 1)
             model = AgglomerativeClustering(linkage=linkage,n_clusters=n_clusters,
             connectivity=connectivity)
             model.fit(X)
             plt.scatter(X[:, 0], X[:, 1], c=model.labels_, cmap=plt.cm.nipy_spectral)
             plt.title('%s linkage' % linkage,fontdict=dict(verticalalignment='top'))
             plt.axis('equal')
             plt.axis('off')
         if connectivity is None:
             plt.suptitle('Without connectivity', size=20)
         else:
```

```
    plt.suptitle('With connectivity', size=20)
    plt.suptitle('Without connectivity: %r' % connectivity is None, size=20)

plt.show()
```

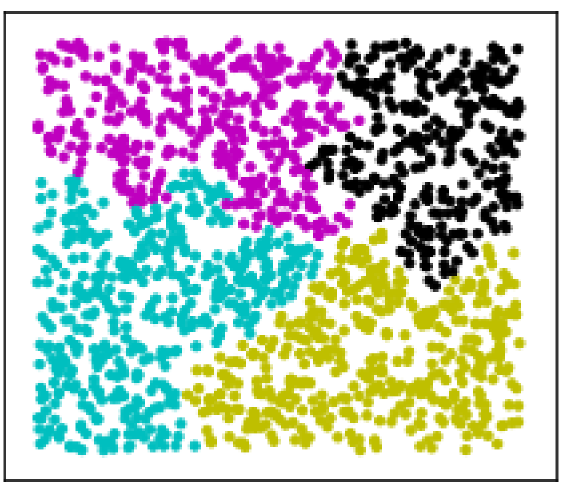
```
<Figure size 864x288 with 0 Axes>
```



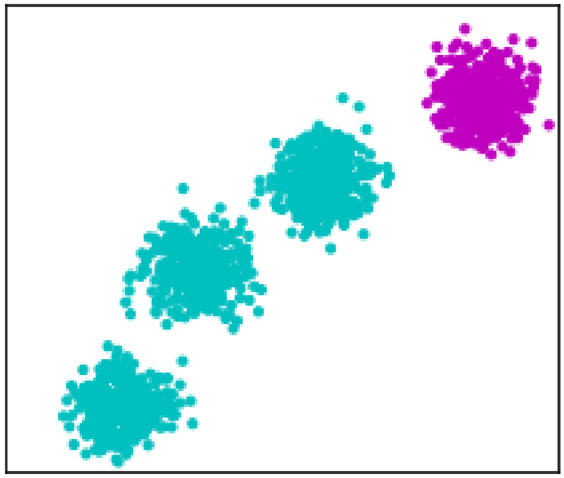

```
In [43]: from sklearn import cluster, datasets

         np.random.seed(0)

         # Generate datasets.
         n_samples = 1500
         no_structure = np.random.rand(n_samples, 2), None
         blobs = datasets.make_blobs(n_samples=n_samples, random_state=8, centers=4)
         noisy_circles = datasets.make_circles(n_samples=n_samples, factor=.5,
         noise=.05)
         noisy_moons = datasets.make_moons(n_samples=n_samples, noise=.05)

         colors = np.array([x for x in 'cmykbgrcmykbgrcmykbgrcmykbgr'])
         colors = np.hstack([colors] * 20)
```

```
In [60]: uniform = np.random.rand(n_samples, 2), None

         plt.figure(figsize=(12, 10))
         plt.subplots_adjust(left=.001, right=.999, bottom=.01, top=.99, wspace=.05, hspace=.01)
         plot_num = 1

         for i, n_clrs in enumerate([2,4]):
             dataset=uniform
             i_dataset=0
             X, y = dataset
```

```
    X = StandardScaler().fit_transform(X)

connectivity = kneighbors_graph(X, n_neighbors=10)
connectivity = 0.5 * (connectivity + connectivity.T)

distances = euclidean_distances(X)

kmeans = cluster.KMeans(n_clusters=n_clrs)
spectral_clustering = cluster.SpectralClustering(n_clusters=n_clrs,affinity="nearest_ne
#eigen_solver='arpack',

average_agglomerative = cluster.AgglomerativeClustering(linkage="average",connectivity=
ward_agglomerative = cluster.AgglomerativeClustering(n_clusters=n_clrs,linkage='ward',

for method, algr in [('KMeans', kmeans),('Spectral Clst.', spectral_clustering),('Avera
    algr.fit(X)

y_pred = algr.labels_.astype(np.int)

plt.subplot(4, 4, plot_num)
if i == 0:
    plt.title(method, size=15)

plt.scatter(X[:, 0], X[:, 1], color=colors[y_pred].tolist(), s=10)

plt.xlim(-2, 2)
plt.ylim(-2, 2)
plt.xticks(())
plt.yticks(())
plot_num += 1
plt.savefig("files/ch07/uniform.png",dpi=300, bbox_inches='tight')
plt.show()
```

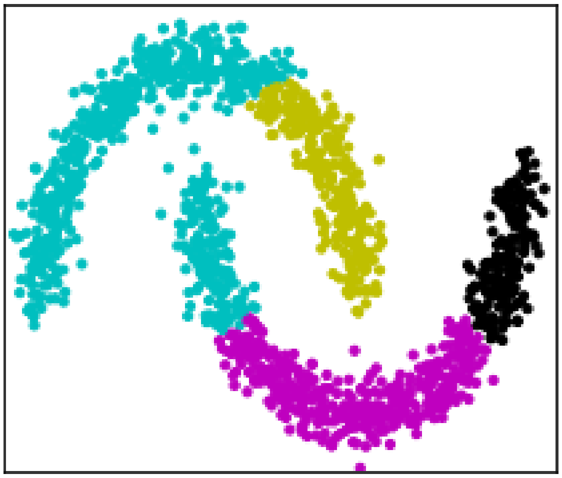


```
In [61]: spheres = datasets.make_blobs(n_samples=n_samples, random_state=3, centers=4)

         plt.figure(figsize=(12, 10))
         plt.subplots_adjust(left=.001, right=.999, bottom=.01, top=.99, wspace=.05, hspace=.01)
         plot_num = 1

         for i, n_clrs in enumerate([4,2]):
             dataset=spheres
             i_dataset=0
             X, y = dataset
             X = StandardScaler().fit_transform(X)

         connectivity = kneighbors_graph(X, n_neighbors=10)
         connectivity = 0.5 * (connectivity + connectivity.T)

         distances = euclidean_distances(X)

         kmeans = cluster.KMeans(n_clusters=n_clrs)
         spectral_clustering = cluster.SpectralClustering(n_clusters=n_clrs,affinity="nearest_ne
         #eigen_solver='arpack',
         average_agglomerative = cluster.AgglomerativeClustering(linkage="average",connectivity=
         ward_agglomerative = cluster.AgglomerativeClustering(n_clusters=n_clrs,linkage='ward',

         for method, algr in [('KMeans', kmeans),('Spectral Clst.', spectral_clustering),('Avera
             algr.fit(X)

         y_pred = algr.labels_.astype(np.int)

         plt.subplot(4, 4, plot_num)
         if i == 0:
             plt.title(method, size=15)

         plt.scatter(X[:, 0], X[:, 1], color=colors[y_pred].tolist(), s=10)

         plt.xlim(-2, 2)
         plt.ylim(-2, 2)
         plt.xticks(())
         plt.yticks(())
         plot_num += 1
         plt.savefig("files/ch07/blobs.png",dpi=300, bbox_inches='tight')
         plt.show()
```



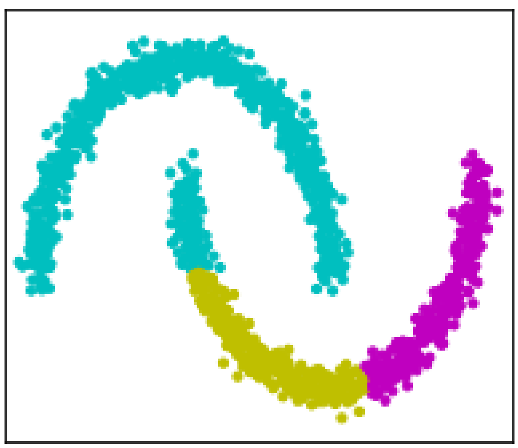
```
In [62]: moons = datasets.make_moons(n_samples=n_samples, noise=.08)

         plt.figure(figsize=(12, 10))
         plt.subplots_adjust(left=.001, right=.999, bottom=.01, top=.99, wspace=.05, hspace=.01)
         plot_num = 1

         for i, n_clrs in enumerate([2,4]):
             dataset=moons
             X, y = dataset
             X = StandardScaler().fit_transform(X)

         connectivity = kneighbors_graph(X, n_neighbors=10)
         connectivity = 0.5 * (connectivity + connectivity.T)
         distances = euclidean_distances(X)

         kmeans = cluster.KMeans(n_clusters=n_clrs)
         spectral_clustering = cluster.SpectralClustering(n_clusters=n_clrs, affinity="nearest_n
         average_agglomerative = cluster.AgglomerativeClustering(linkage="average",connectivity=
         ward_agglomerative = cluster.AgglomerativeClustering(n_clusters=n_clrs,linkage='ward',

         for method, algr in [('KMeans', kmeans),('Spectral Clst.', spectral_clustering),('Avera
             algr.fit(X)

         y_pred = algr.labels_.astype(np.int)

         plt.subplot(4, 4, plot_num)
         if i == 0:
             plt.title(method, size=15)

         plt.scatter(X[:, 0], X[:, 1], color=colors[y_pred].tolist(), s=10)
```

```
plt.xlim(-2, 2)
plt.ylim(-2, 2)
plt.xticks(())
plt.yticks(())
plot_num += 1
plt.savefig("files/ch07/moons.png" , bbox_inches='tight')
plt.show()
```



```
In [63]: plt.figure(figsize=(11, 9.5))
         plt.subplots_adjust(left=.001, right=.999, bottom=.001, top=.96, wspace=.05, hspace=.01
         plot_num = 1

         for i_dataset, dataset in enumerate([blobs,
             no_structure, noisy_circles, noisy_moons]):
             X, y = dataset
             X = StandardScaler().fit_transform(X)

         connectivity = kneighbors_graph(X, n_neighbors=10)
         connectivity = 0.5 * (connectivity + connectivity.T)

         distances = euclidean_distances(X)

         means = cluster.KMeans(n_clusters=4)
         spectral = cluster.SpectralClustering(n_clusters=3, eigen_solver='arpack',affinity="nea
         average_linkage = cluster.AgglomerativeClustering(linkage="average",affinity="cityblock
         ward = cluster.AgglomerativeClustering(n_clusters=3,linkage='ward', connectivity=connec

         for name, algorithm in [('KMeans', means),('SpectrClust', spectral),('AgglomClust (aver
             algorithm.fit(X)
```

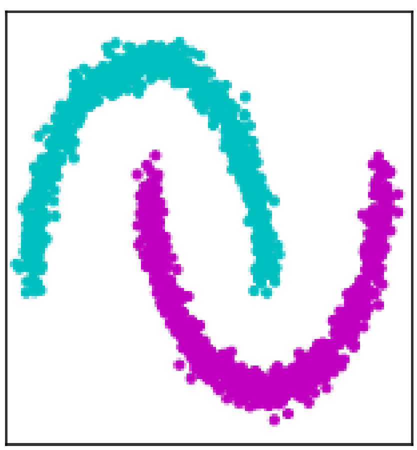
```
        if hasattr(algorithm, 'labels_'):
            y_pred = algorithm.labels_.astype(np.int)
        else:
            y_pred = algorithm.predict(X)

        plt.subplot(4, 4, plot_num)
        if i_dataset == 0:
            plt.title(name, size=18)

        plt.scatter(X[:, 0], X[:, 1], color=colors[y_pred].tolist(), s=10)

        plt.xlim(-2, 2)
        plt.ylim(-2, 2)
        plt.xticks(())
        plt.yticks(())
        plot_num += 1

        plt.show()
```



The code provided by scikit-learn for comparing the different clustering techniques when k=2 is as follows:

```
In [64]: plt.figure(figsize=(11, 9.5))
        plt.subplots_adjust(left=.001, right=.999, bottom=.001, top=.96, wspace=.05, hspace=.01

        plot_num = 1
        for i_dataset, dataset in enumerate([blobs, no_structure, noisy_circles,noisy_moons]):
            X, y = dataset

        X = StandardScaler().fit_transform(X)
```

```
connectivity = kneighbors_graph(X, n_neighbors=10)
connectivity = 0.5 * (connectivity + connectivity.T)

distances = euclidean_distances(X)

means = cluster.KMeans(n_clusters=2)
spectral = cluster.SpectralClustering(n_clusters=2, eigen_solver='arpack',affinity="nea
average_linkage = cluster.AgglomerativeClustering(linkage="average",affinity="cityblock
complete_linkage = cluster.AgglomerativeClustering(linkage="complete",affinity="cityblo
ward_linkage = cluster.AgglomerativeClustering(n_clusters=2,linkage='ward', connectivit

for name, algorithm in [('KMeans', means),('AgglomClust (average)', average_linkage),('
    algorithm.fit(X)
if hasattr(algorithm, 'labels_'):
    y_pred = algorithm.labels_.astype(np.int)
else:
    y_pred = algorithm.predict(X)

plt.subplot(4, 5, plot_num)
plt.scatter(X[:, 0], X[:, 1], color=colors[y_pred].tolist(), s=10)

plt.xlim(-2, 2)
plt.ylim(-2, 2)
plt.xticks(())
plt.yticks(())
plot_num += 1

plt.show()
```



CASE STUDY: EUROSTAT data analysis **Applying clustering to analyze countries according to their education resourses**

In order to illustrate the clustering on a real dataset, we will analyze the indicators on education finance data among the European member states, provided by the Eurostat data bank2. The data is organized by year (TIME): [2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011] and country (GEO): ['Albania', 'Austria', 'Belgium', 'Bulgaria', 'Croatia', 'Cyprus', 'Czech Republic', 'Denmark', 'Estonia', 'Euro area (13 countries)', 'Euro area (15 countries)', 'European Union (25 countries)', 'European Union (27 countries)', 'Finland', 'Former Yugoslav Republic of Macedonia, the', 'France', 'Germany (until 1990 former territory of the FRG)', 'Greece', 'Hungary', 'Iceland', 'Ireland', 'Italy', 'Japan', 'Latvia', 'Liechtenstein', 'Lithuania', 'Luxembourg', 'Malta', 'Netherlands', 'Norway', 'Poland', 'Portugal', 'Romania', 'Slovakia', 'Slovenia', 'Spain', 'Sweden', 'Switzerland', 'Turkey', 'United Kingdom', 'United States']. Twelve indicators (INDICED) on education finance with their values (Value) are given like:-

1. Expenditure on educational institutions from private sources as % of Gross Domestic Product (GDP), for all levels of education combined;
2. Expenditure on educational institutions from public sources as % of GDP, for all levels of government combined,
3. Expenditure on educational institutions from public sources as % of total public expenditure, for all levels of education combined,
4. Public subsidies to the private sector as % of GDP, for all levels of education combined,
5. Public subsidies to the private sector as % of total public expenditure, for all levels of education combined, etc. We can store in a table the 12 indicators for a given year (e.g. 2010).

Let us start having a look at the data.

```
In [65]: #Read and check the dataset downloaded from the EuroStat

         import pandas as pd
         import numpy as np

         from sklearn.preprocessing import StandardScaler
         from sklearn import cluster

         edu=pd.read_csv('./files/ch07/educ_figdp_1_Data.csv',na_values=':')
         edu.head()
```

```
Out[65]:    TIME                            GEO  \
         0  2002  European Union (27 countries)
         1  2002  European Union (27 countries)
         2  2002  European Union (27 countries)
         3  2002  European Union (27 countries)
         4  2002  European Union (27 countries)

                                                   INDIC_ED  Value
         0  Total public expenditure on education as % of ...   5.10
         1  Total public expenditure on education as % of ...   1.14
         2  Total public expenditure on education as % of ...   2.32
         3  Total public expenditure on education as % of ...   1.15
         4  Total public expenditure on education as % of ...   0.50
```

```
In [66]: edu.tail()
```

```
Out[66]:       TIME    GEO                                           INDIC_ED  Value
         4915  2011  Japan  Total public expenditure on education as % of ...    NaN
         4916  2011  Japan  Expenditure on educational institutions from p...    NaN
         4917  2011  Japan  Public subsidies to the private sector as % of...    NaN
         4918  2011  Japan  Expenditure on educational institutions from p...   1.56
         4919  2011  Japan  Total public expenditure on education as % of ...   3.67
```

Data in CSV and databases are often organised in what is called stacked or record formats. In our case for each year ("TIME") and country ("GEO") of the EU as well as some reference countries such as Japan and United States, we have twelve indicators ("INDIC_ED") on education finance with their values ("Value"). Let us reshape the table into a feature vector style data set.

To the process of reshaping stacked data into a table is sometimes called pivoting.

```
In [67]: #Pivot table in order to get a nice feature vector representation with dual indexing by
         pivedu=pd.pivot_table(edu, values='Value', index=['TIME', 'GEO'], columns=['INDIC_ED'])
         pivedu.head()
```

```
Out[67]: INDIC_ED      Expenditure on educational institutions from private sources as % of GDP
         TIME GEO
         2002 Albania                                                   NaN
              Austria                                                  0.38
              Belgium                                                  0.36
              Bulgaria                                                 0.67
              Croatia                                                  0.13

         INDIC_ED      Expenditure on educational institutions from public sources as % of GDP,
         TIME GEO
         2002 Albania                                                   NaN
              Austria                                                  5.30
              Belgium                                                  5.80
              Bulgaria                                                 3.75
              Croatia                                                  3.71

         INDIC_ED      Expenditure on educational institutions from public sources as % of tota
         TIME GEO
         2002 Albania                                                   NaN
              Austria                                                 10.46
              Belgium                                                 11.65
              Bulgaria                                                 9.49
              Croatia                                                   NaN

         INDIC_ED      Public subsidies to the private sector as % of GDP, for all levels of ed
         TIME GEO
         2002 Albania                                                   NaN
              Austria                                                  0.37
              Belgium                                                  0.29
              Bulgaria                                                 0.18
```

```
     Croatia                                               NaN

INDIC_ED       Public subsidies to the private sector as % of total public expenditure,
TIME GEO
2002 Albania                                               NaN
     Austria                                              0.74
     Belgium                                              0.58
     Bulgaria                                             0.46
     Croatia                                               NaN

INDIC_ED       Total public expenditure on education as % of GDP, at pre-primary level
TIME GEO
2002 Albania                                               NaN
     Austria                                              0.63
     Belgium                                              0.70
     Bulgaria                                             0.67
     Croatia                                              0.44

INDIC_ED       Total public expenditure on education as % of GDP, at primary level of e
TIME GEO
2002 Albania                                               NaN
     Austria                                              1.12
     Belgium                                              1.36
     Bulgaria                                             0.73
     Croatia                                              1.81

INDIC_ED       Total public expenditure on education as % of GDP, at secondary level of
TIME GEO
2002 Albania                                               NaN
     Austria                                              2.64
     Belgium                                              2.71
     Bulgaria                                             1.72
     Croatia                                              0.88

INDIC_ED       Total public expenditure on education as % of GDP, at tertiary level of
TIME GEO
2002 Albania                                               NaN
     Austria                                              1.28
     Belgium                                              1.32
     Bulgaria                                             0.81
     Croatia                                              0.59

INDIC_ED       Total public expenditure on education as % of GDP, for all levels of edu
TIME GEO
2002 Albania                                               NaN
     Austria                                              5.68
     Belgium                                              6.09
     Bulgaria                                             3.94
```

```
     Croatia                                                  3.71

INDIC_ED      Total public expenditure on education as % of gross national income, for
TIME GEO
2002 Albania                                                   NaN
     Austria                                                  5.75
     Belgium                                                  6.01
     Bulgaria                                                 3.85
     Croatia                                                  3.77

INDIC_ED      Total public expenditure on education as % of total public expenditure,
TIME GEO
2002 Albania                                                   NaN
     Austria                                                 11.20
     Belgium                                                 12.23
     Bulgaria                                                 9.95
     Croatia                                                   NaN
```

In [68]:
```python
print ('Let us check the two indices:\n')
print ('\nPrimary index (TIME): \n' + str(pivedu.index.levels[0].tolist()))
print ('\nSecondary index (GEO): \n' + str(pivedu.index.levels[1].tolist()))
```

```
Let us check the two indices:


Primary index (TIME):
[2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011]

Secondary index (GEO):
['Albania', 'Austria', 'Belgium', 'Bulgaria', 'Croatia', 'Cyprus', 'Czech Republic', 'Denmark',
```

In [69]:
```python
#Extract 2010 set of values
edu2010=pivedu.ix[2010]
edu2010.head()
```

```
/home/nbuser/anaconda3_501/lib/python3.6/site-packages/ipykernel/__main__.py:2: DeprecationWarni
.ix is deprecated. Please use
.loc for label based indexing or
.iloc for positional indexing

See the documentation here:
http://pandas.pydata.org/pandas-docs/stable/indexing.html#ix-indexer-is-deprecated
  from ipykernel import kernelapp as app
```

Out[69]:
```
INDIC_ED  Expenditure on educational institutions from private sources as % of GDP, for
GEO
Albania                                                    NaN
```

```
Austria                                               0.52
Belgium                                               0.34
Bulgaria                                              0.63
Croatia                                               0.26

INDIC_ED  Expenditure on educational institutions from public sources as % of GDP, for
GEO
Albania                                                NaN
Austria                                               5.25
Belgium                                               6.25
Bulgaria                                              3.35
Croatia                                               4.24

INDIC_ED  Expenditure on educational institutions from public sources as % of total pub
GEO
Albania                                                NaN
Austria                                               9.98
Belgium                                              11.90
Bulgaria                                              8.96
Croatia                                                NaN

INDIC_ED  Public subsidies to the private sector as % of GDP, for all levels of educati
GEO
Albania                                                NaN
Austria                                               0.64
Belgium                                               0.32
Bulgaria                                              0.74
Croatia                                               0.03

INDIC_ED  Public subsidies to the private sector as % of total public expenditure, for
GEO
Albania                                                NaN
Austria                                               1.22
Belgium                                               0.61
Bulgaria                                              1.99
Croatia                                                NaN

INDIC_ED  Total public expenditure on education as % of GDP, at pre-primary level of ed
GEO
Albania                                                NaN
Austria                                               0.61
Belgium                                               0.78
Bulgaria                                              0.92
Croatia                                               0.65

INDIC_ED  Total public expenditure on education as % of GDP, at primary level of educat
GEO
Albania                                                NaN
```

```
Austria                                                       1.01
Belgium                                                       1.54
Bulgaria                                                      0.80
Croatia                                                       1.87

INDIC_ED  Total public expenditure on education as % of GDP, at secondary level of educ
GEO
Albania                                                        NaN
Austria                                                       2.64
Belgium                                                       2.79
Bulgaria                                                      1.76
Croatia                                                       0.97

INDIC_ED  Total public expenditure on education as % of GDP, at tertiary level of educa
GEO
Albania                                                        NaN
Austria                                                       1.63
Belgium                                                       1.46
Bulgaria                                                      0.61
Croatia                                                       0.78

INDIC_ED  Total public expenditure on education as % of GDP, for all levels of educatio
GEO
Albania                                                        NaN
Austria                                                       5.89
Belgium                                                       6.57
Bulgaria                                                      4.10
Croatia                                                       4.27

INDIC_ED  Total public expenditure on education as % of gross national income, for all
GEO
Albania                                                        NaN
Austria                                                       5.90
Belgium                                                       6.44
Bulgaria                                                      4.18
Croatia                                                       4.42

INDIC_ED  Total public expenditure on education as % of total public expenditure, for a
GEO
Albania                                                        NaN
Austria                                                      11.20
Belgium                                                      12.51
Bulgaria                                                     10.95
Croatia                                                        NaN
```

Let us clean and store the names of the features and the countries.

In [70]:
```
#Store column names and clear them for better handling. Do the same with countries
edu2010 = edu2010.rename(index={'Euro area (13 countries)': 'EU13',
```

```
'Euro area (15 countries)': 'EU15',
'European Union (25 countries)': 'EU25',
'European Union (27 countries)': 'EU27',
'Former Yugoslav Republic of Macedonia, the': 'Macedonia',
'Germany (until 1990 former territory of the FRG)': 'Germany'
})
features = edu2010.columns.tolist()

countries = edu2010.index.tolist()

edu2010.columns=range(12)
edu2010.head()
```

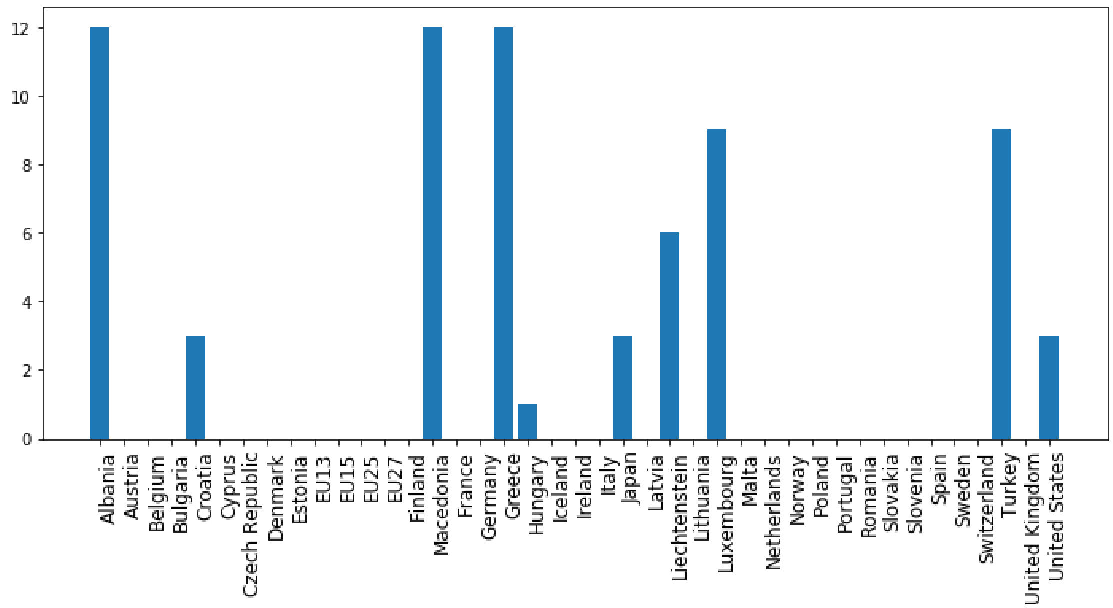
Out[70]:

```
             0     1      2     3     4     5     6     7     8     9    10  \
GEO
Albania    NaN   NaN    NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN
Austria   0.52  5.25   9.98  0.64  1.22  0.61  1.01  2.64  1.63  5.89  5.90
Belgium   0.34  6.25  11.90  0.32  0.61  0.78  1.54  2.79  1.46  6.57  6.44
Bulgaria  0.63  3.35   8.96  0.74  1.99  0.92  0.80  1.76  0.61  4.10  4.18
Croatia   0.26  4.24    NaN  0.03   NaN  0.65  1.87  0.97  0.78  4.27  4.42

              11
GEO
Albania      NaN
Austria    11.20
Belgium    12.51
Bulgaria   10.95
Croatia      NaN
```

As we can observe, this is not a clean data set, there are missing values. Some countries may not collect or have access to some indicators and there are countries without any indicators. Let us display this effect.

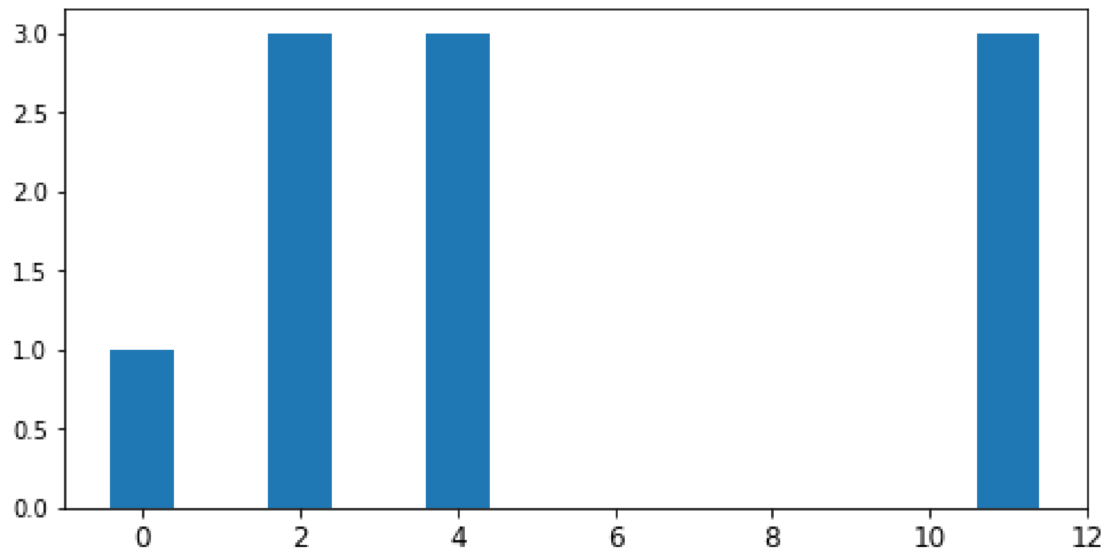
In [71]:

```
#Check what is going on in the NaN data
nan_countries=np.sum(np.where(edu2010.isnull(),1,0),axis=1)
plt.bar(np.arange(nan_countries.shape[0]),nan_countries)
plt.xticks(np.arange(nan_countries.shape[0]),countries,rotation=90,horizontalalignment=
fontsize=12)
fig = plt.gcf()
fig.set_size_inches((12,5))
```




```
In [72]: #Remove non info countries
         wrk_countries = nan_countries<4

         educlean=edu2010.ix[wrk_countries] #.ix - Construct an open mesh from multiple sequence

         #Let us check the features we have
         na_features = np.sum(np.where(educlean.isnull(),1,0),axis=0)
         print (na_features)

         plt.bar(np.arange(na_features.shape[0]),na_features)
         plt.xticks(fontsize=12)
         fig = plt.gcf()
         fig.set_size_inches((8,4))
```

```
/home/nbuser/anaconda3_501/lib/python3.6/site-packages/ipykernel/__main__.py:4: DeprecationWarni
.ix is deprecated. Please use
.loc for label based indexing or
.iloc for positional indexing

See the documentation here:
http://pandas.pydata.org/pandas-docs/stable/indexing.html#ix-indexer-is-deprecated


[1 0 3 0 3 0 0 0 0 0 0 3]
```



There are four features with missing data. At this point we can proceed in two ways: - Fill in the features with some non-informative, non-biasing data. - Drop the features with missing values.

If we have many features and only a few have missing values then it is not much harmful to drop them. However, if missing values are spread across the features, we have to eventually deal with them. In our case, both options seem reasonable, so we will proceed with both at the same time.

```
In [73]: #Option A fills those features with some value, at risk of extracting wrong information
         #Constant filling : edufill0=educlean.fillna(0)
         edufill=educlean.fillna(educlean.mean())
         print ('Filled in data shape: ' + str(edufill.shape))

         #Option B drops those features
         edudrop=educlean.dropna(axis=1)
         #dropna: Return object with labels on given axis omitted where alternately any or
         # all of the data are missing
         print ('Drop data shape: ' + str(edudrop.shape))
```

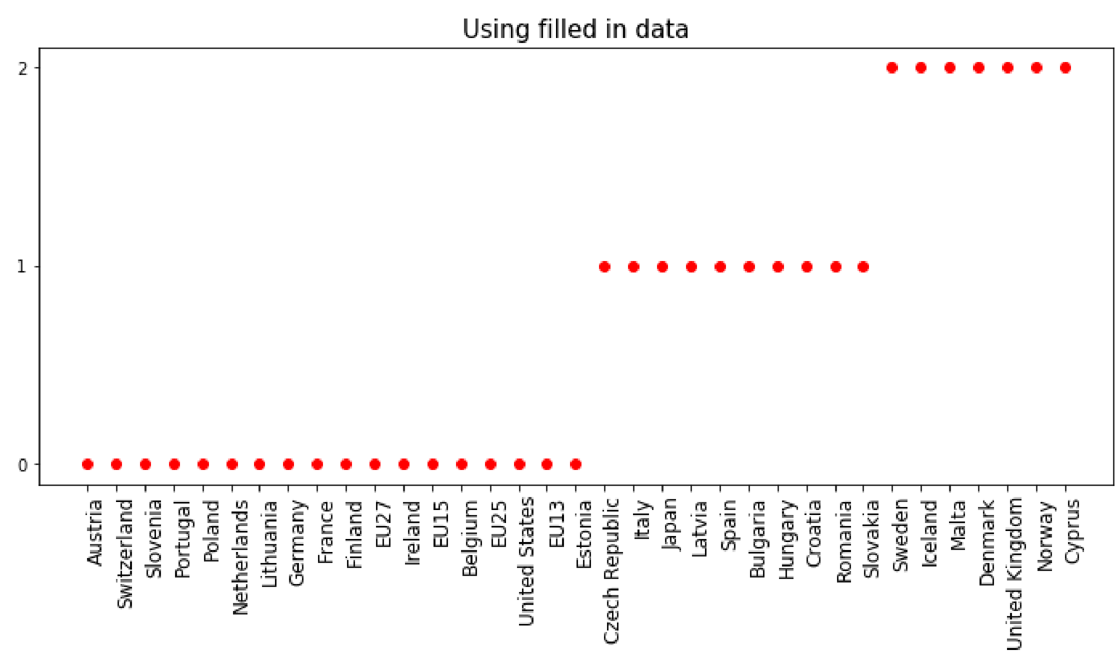
```
Filled in data shape: (35, 12)
Drop data shape: (35, 8)
```

Let us now apply a K-means clustering technique on this data in order to partition the countries according to their investment in education and check their profiles.

```
In [74]: scaler = StandardScaler() #Standardize features by removing the mean and scaling to uni

         X_train_fill = edufill.values
```

```
X_train_fill = scaler.fit_transform(X_train_fill)

clf = cluster.KMeans(init='k-means++', n_clusters=3, random_state=42)

clf.fit(X_train_fill) #Compute k-means clustering.

y_pred_fill = clf.predict(X_train_fill)
#Predict the closest cluster each sample in X belongs to.

idx=y_pred_fill.argsort()
```

```
In [75]: plt.plot(np.arange(35),y_pred_fill[idx],'ro')
         wrk_countries_names = [countries[i] for i,item in enumerate(wrk_countries) if item ]

         plt.xticks(np.arange(len(wrk_countries_names)),[wrk_countries_names[i] for i in idx],
         rotation=90,horizontalalignment='left',fontsize=12)
         plt.title('Using filled in data', size=15)
         plt.yticks([0,1,2])
         fig = plt.gcf()

         fig.set_size_inches((12,5))
```

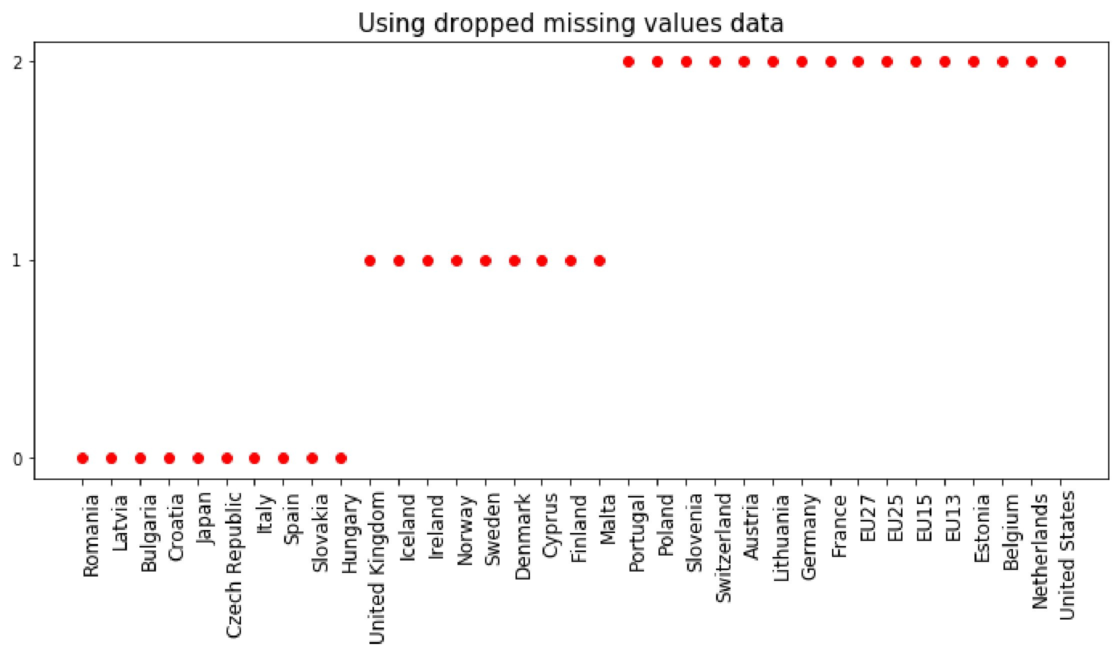


Applying clustering with dataset with dropped missing values

```
In [76]: X_train_drop = edudrop.values
         X_train_drop = scaler.fit_transform(X_train_drop)
```

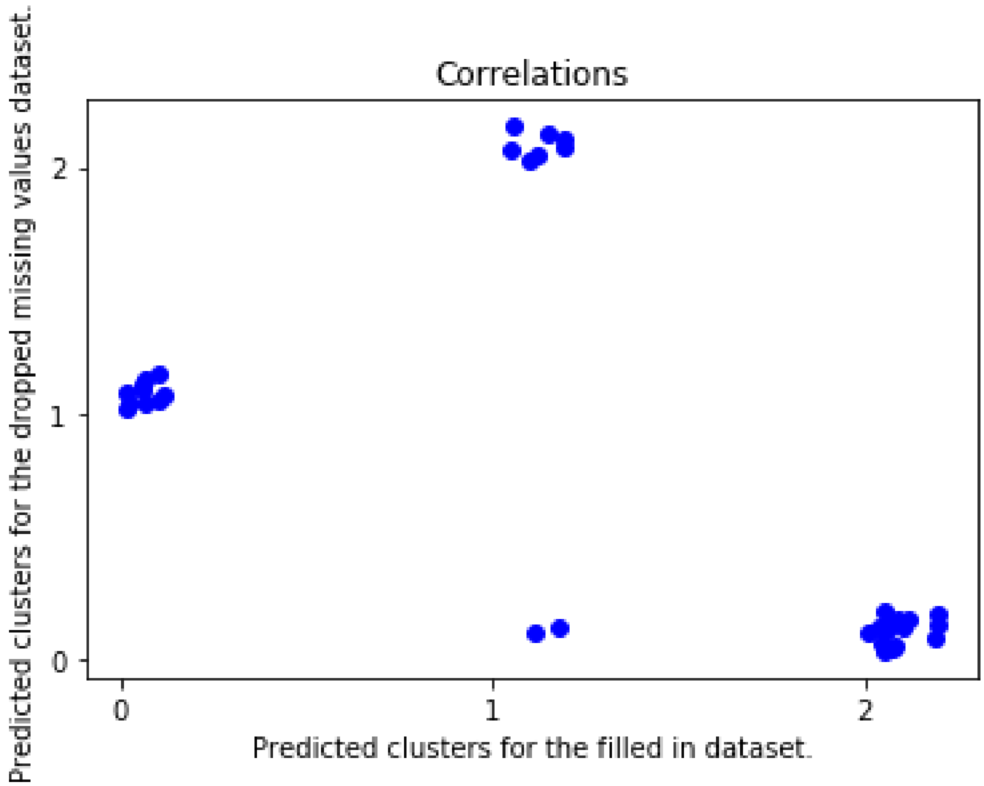
```
clf.fit(X_train_drop) #Compute k-means clustering.
y_pred_drop = clf.predict(X_train_drop) #Predict the closest cluster of each sample in
```

```
In [77]: idx=y_pred_drop.argsort()
         plt.plot(np.arange(35),y_pred_drop[idx],'ro')
         wrk_countries_names = [countries[i] for i,item in enumerate(wrk_countries) if item ]

         plt.xticks(np.arange(len(wrk_countries_names)),[wrk_countries_names[i] for i in idx],
         rotation=90,horizontalalignment='left',fontsize=12)
         plt.title('Using dropped missing values data',size=15)
         fig = plt.gcf()
         plt.yticks([0,1,2])
         fig.set_size_inches((12,5))
```



We have sorted the data for better visualization. At a simple glance we can see that both partitions can be different. We can better check this effect plotting the clusters values of one technique against the other.

```
In [78]: plt.plot(y_pred_drop+0.2*np.random.rand(35),y_pred_fill+0.2*np.random.rand(35),'bo')
         plt.xlabel('Predicted clusters for the filled in dataset.')
         plt.ylabel('Predicted clusters for the dropped missing values dataset.')
         plt.title('Correlations')
         plt.xticks([0,1,2])
         plt.yticks([0,1,2])
         plt.savefig("files/ch07/correlationkmeans.png",dpi=300, bbox_inches='tight')
```



Let us check the list of countries in both methods. Note that we should not consider the cluster value, since it is irrelevant.

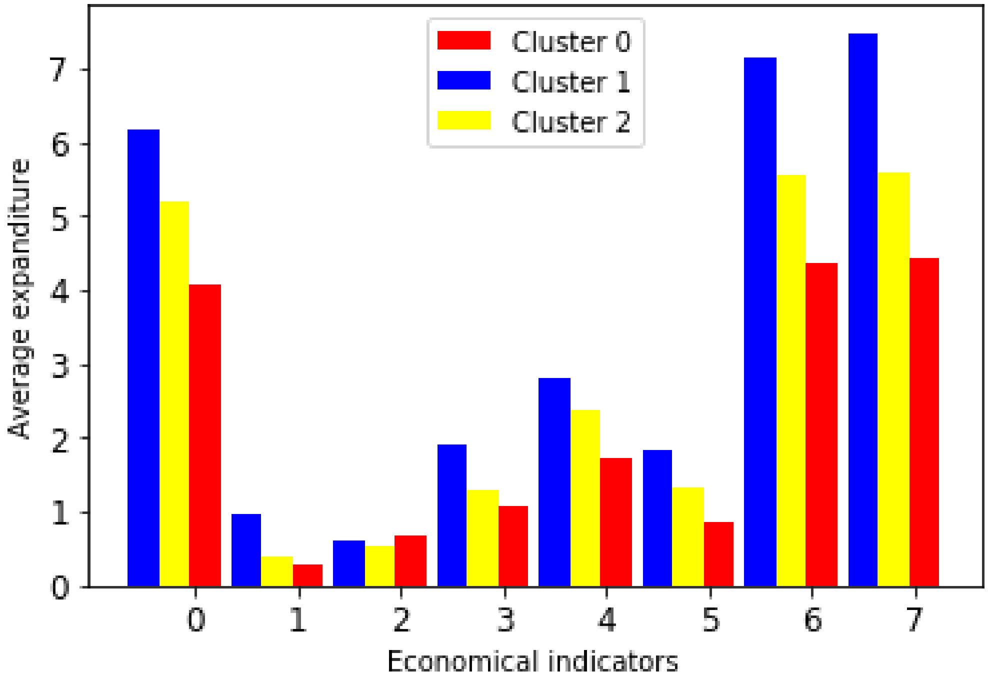
```
In [82]: print ('Cluster 0: \n' + str([wrk_countries_names[i] for i,item in enumerate(y_pred_fil
         print ('Cluster 0: \n' + str([wrk_countries_names[i] for i,item in enumerate(y_pred_dro
         print ('\n')
         print ('Cluster 1: \n' + str([wrk_countries_names[i] for i,item in enumerate(y_pred_fil
         print ('Cluster 1: \n' + str([wrk_countries_names[i] for i,item in enumerate(y_pred_dro
         print ('\n')
         print ('Cluster 2: \n' + str([wrk_countries_names[i] for i,item in enumerate(y_pred_fil
         print ('Cluster 2: \n' + str([wrk_countries_names[i] for i,item in enumerate(y_pred_dro
         print ('\n')
```

```
Cluster 0:
['Austria', 'Belgium', 'Estonia', 'EU13', 'EU15', 'EU25', 'EU27', 'Finland', 'France', 'Germany'
Cluster 0:
['Bulgaria', 'Croatia', 'Czech Republic', 'Hungary', 'Italy', 'Japan', 'Latvia', 'Romania', 'Slo


Cluster 1:
['Bulgaria', 'Croatia', 'Czech Republic', 'Hungary', 'Italy', 'Japan', 'Latvia', 'Romania', 'Slo
Cluster 1:
['Cyprus', 'Denmark', 'Finland', 'Iceland', 'Ireland', 'Malta', 'Norway', 'Sweden', 'United King
```

```
Cluster 2:
['Cyprus', 'Denmark', 'Iceland', 'Malta', 'Norway', 'Sweden', 'United Kingdom']
Cluster 2:
['Austria', 'Belgium', 'Estonia', 'EU13', 'EU15', 'EU25', 'EU27', 'France', 'Germany', 'Lithuani
```

Let us check the profile of the clusters by looking at the centroids:

```
In [83]: width=0.3
         p1 = plt.bar(np.arange(8),scaler.inverse_transform(clf.cluster_centers_[1]),width,color
         # Scale back the data to the original representation
         p2 = plt.bar(np.arange(8)+width,scaler.inverse_transform(clf.cluster_centers_[2]),width
         p0 = plt.bar(np.arange(8)+2*width,scaler.inverse_transform(clf.cluster_centers_[0]),wid

         plt.legend( (p0[0], p1[0], p2[0]), ('Cluster 0', 'Cluster 1', 'Cluster 2') ,loc=9)
         plt.xticks(np.arange(8) + 0.5, np.arange(8),size=12)
         plt.yticks(size=12)
         plt.xlabel('Economical indicators')
         plt.ylabel('Average expanditure')
         fig = plt.gcf()

         plt.savefig("files/ch07/clusterexpenditure.png",dpi=300, bbox_inches='tight')
```

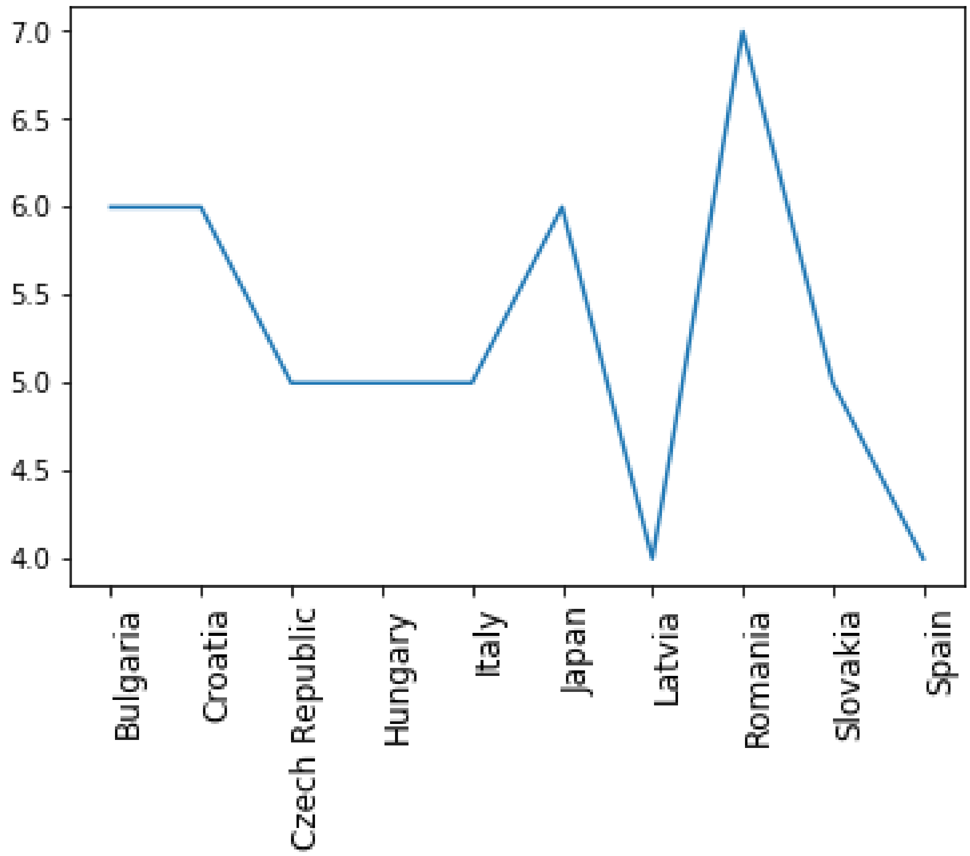


It looks like cluster “1“ spends more on education while cluster “0“ is the one with less resources on education. What about Spain?

Let us refine a little bit more cluster “0“ and check how close are members from this cluster to cluster “1“. This may give us a hint on a possible ordering.

```
In [85]: from scipy.spatial import distance
         p = distance.cdist(X_train_drop[y_pred_drop==0,:],[clf.cluster_centers_[1]],'euclidean'
         #the distance of the elements of cluster 0 to the center of cluster 1

         fx = np.vectorize(np.int)

         plt.plot(np.arange(p.shape[0]),fx(p))

         wrk_countries_names = [countries[i] for i,item in enumerate(wrk_countries) if item ]
         zero_countries_names = [wrk_countries_names[i] for i,item in enumerate(y_pred_drop) if
         plt.xticks(np.arange(len(zero_countries_names)),zero_countries_names,rotation=90,horizo
```

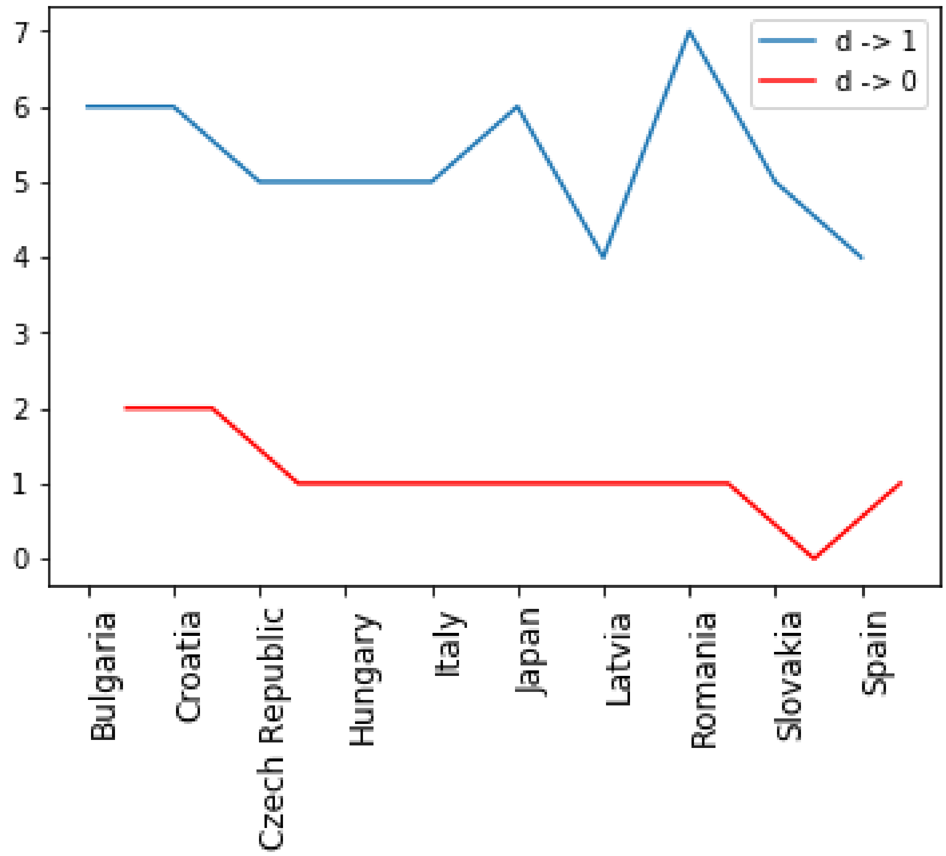
```
Out[85]: ([<matplotlib.axis.XTick at 0x7f8f5fea17b8>,
           <matplotlib.axis.XTick at 0x7f8f5fea10f0>,
           <matplotlib.axis.XTick at 0x7f8f5ff88b00>,
           <matplotlib.axis.XTick at 0x7f8f5feb6940>,
           <matplotlib.axis.XTick at 0x7f8f5feb6e10>,
           <matplotlib.axis.XTick at 0x7f8f5febd320>,
           <matplotlib.axis.XTick at 0x7f8f5febd828>,
           <matplotlib.axis.XTick at 0x7f8f5febdd30>,
           <matplotlib.axis.XTick at 0x7f8f5fe44278>,
           <matplotlib.axis.XTick at 0x7f8f5fe44780>],
          <a list of 10 Text xticklabel objects>)
```



Well, it seems that Spain belongs to cluster "0", it is the closest to change to a policy in the lines of the other clusters.

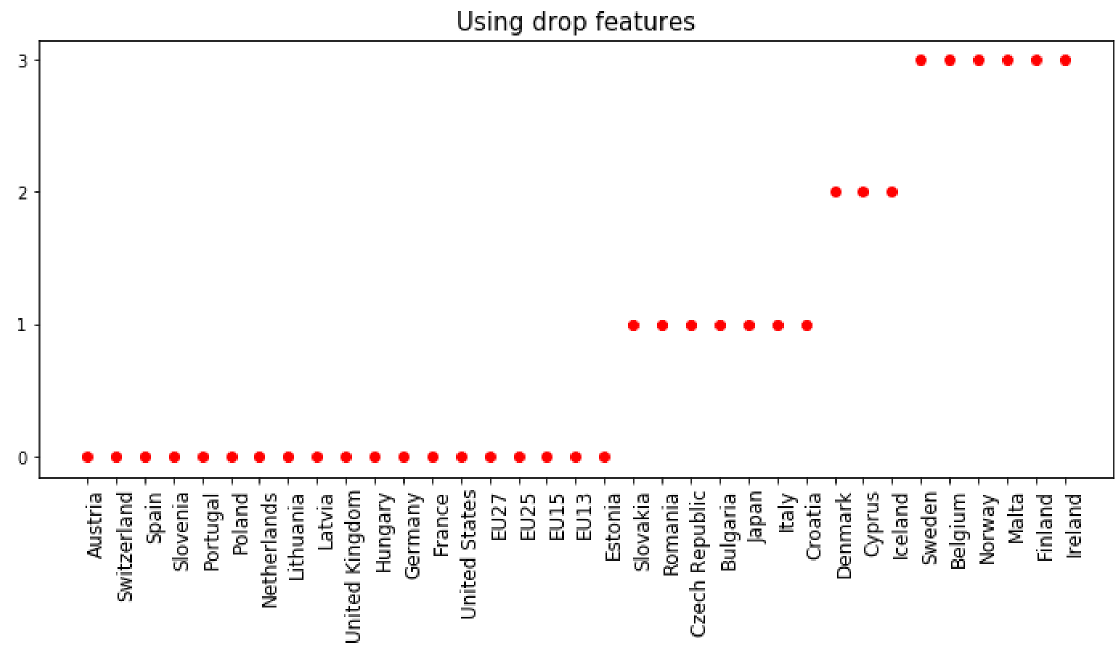
Additionally, we can also check the distance to the centroid of cluster "0".

In [87]:
```
from scipy.spatial import distance
p = distance.cdist(X_train_drop[y_pred_drop==0,:],[clf.cluster_centers_[1]],'euclidean'
pown = distance.cdist(X_train_drop[y_pred_drop==0,:],[clf.cluster_centers_[0]],'euclide

width=0.45
p0=plt.plot(np.arange(p.shape[0]),fx(p),width)
p1=plt.plot(np.arange(p.shape[0])+width,fx(pown),width,color = 'red')

wrk_countries_names = [countries[i] for i,item in enumerate(wrk_countries) if item ]
zero_countries_names = [wrk_countries_names[i] for i,item in enumerate(y_pred_drop) if
plt.xticks(np.arange(len(zero_countries_names)),zero_countries_names,rotation=90,
horizontalalignment='left',fontsize=12)
plt.legend( (p0[0], p1[0]), ('d -> 1', 'd -> 0') ,loc=1)
plt.savefig("files/ch07/dist2cluster01.png",dpi=300, bbox_inches='tight')
```

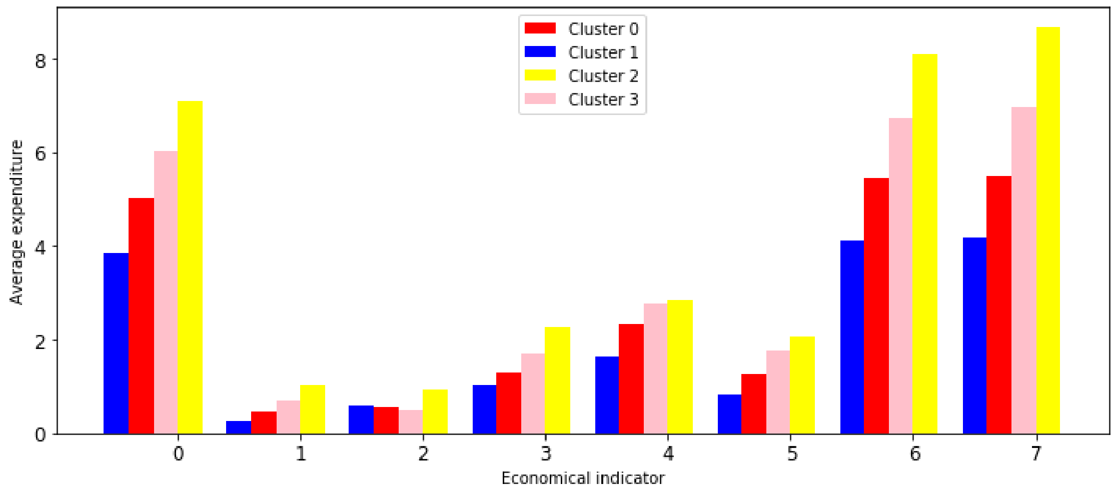


Let us redo the clustering with K=4 and see what we can conclude.

```
In [88]: X_train = edudrop.values
         clf = cluster.KMeans(init='k-means++', n_clusters=4, random_state=0)
         clf.fit(X_train)
         y_pred = clf.predict(X_train)

         idx=y_pred.argsort()
         plt.plot(np.arange(35),y_pred[idx],'ro')
         wrk_countries_names = [countries[i] for i,item in enumerate(wrk_countries) if item ]

         plt.xticks(np.arange(len(wrk_countries_names)),[wrk_countries_names[i] for i in idx],ro
         plt.title('Using drop features',size=15)
         plt.yticks([0,1,2,3])
         fig = plt.gcf()
         fig.set_size_inches((12,5))
```



In [89]:
```
width=0.2
p0 = plt.bar(np.arange(8)+1*width,clf.cluster_centers_[0],width,color='r')
p1 = plt.bar(np.arange(8),clf.cluster_centers_[1],width,color='b')
p2 = plt.bar(np.arange(8)+3*width,clf.cluster_centers_[2],width,color='yellow')
p3 = plt.bar(np.arange(8)+2*width,clf.cluster_centers_[3],width,color='pink')

plt.legend( (p0[0], p1[0], p2[0], p3[0]), ('Cluster 0', 'Cluster 1', 'Cluster 2','Clust
plt.xticks(np.arange(8) + 0.5, np.arange(8),size=12)
plt.yticks(size=12)
plt.xlabel('Economical indicator')
plt.ylabel('Average expenditure')
fig = plt.gcf()
fig.set_size_inches((12,5))
plt.savefig("files/ch07/distances4clusters.png",dpi=300, bbox_inches='tight')
```



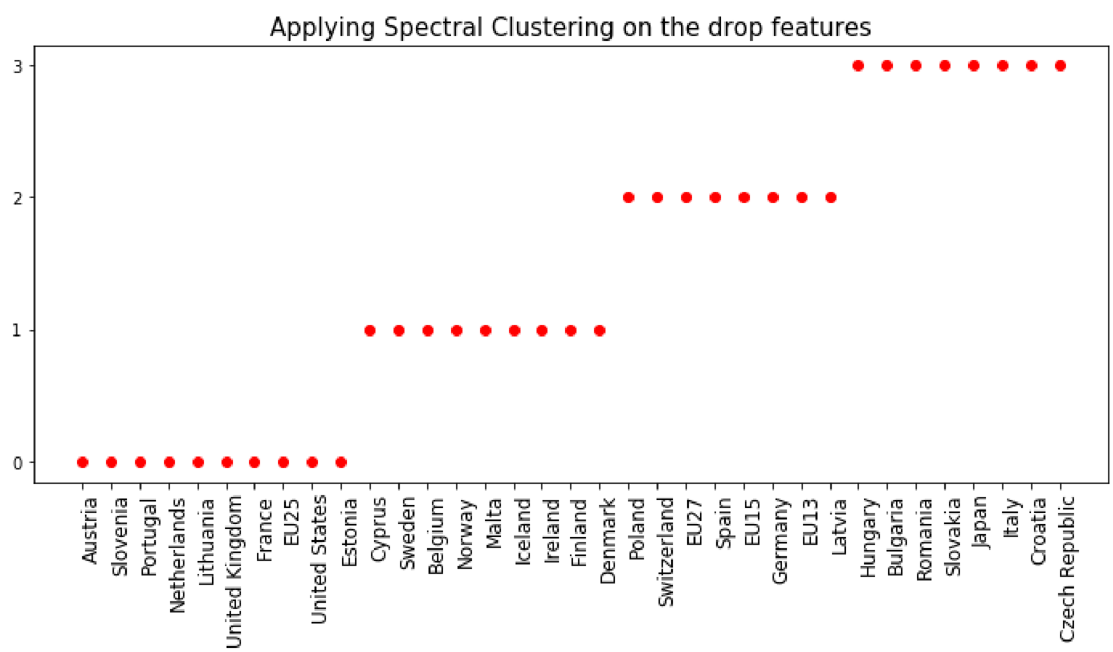
Spain is still in cluster “0“. But as we observed in our previous clustering it was very close to changing cluster. This time cluster “0“ includes the averages values for the EU members. Just for the sake of completeness, let us write down the name of the countries in the clusters.

```
In [90]: print ('Cluster 0: \n' + str([wrk_countries_names[i] for i,item in enumerate(y_pred) if
         print ('Cluster 1: \n' + str([wrk_countries_names[i] for i,item in enumerate(y_pred) if
         print ('Cluster 2: \n' + str([wrk_countries_names[i] for i,item in enumerate(y_pred) if
         print ('Cluster 3: \n' + str([wrk_countries_names[i] for i,item in enumerate(y_pred) if

         #Save data for future use.
         import pickle
         ofname = open('edu2010.pkl', 'wb')
         s = pickle.dump([edu2010, wrk_countries_names,y_pred ],ofname)
         ofname.close()

Cluster 0:
['Austria', 'Estonia', 'EU13', 'EU15', 'EU25', 'EU27', 'France', 'Germany', 'Hungary', 'Latvia',
Cluster 1:
['Bulgaria', 'Croatia', 'Czech Republic', 'Italy', 'Japan', 'Romania', 'Slovakia']
Cluster 2:
['Cyprus', 'Denmark', 'Iceland']
Cluster 3:
['Belgium', 'Finland', 'Ireland', 'Malta', 'Norway', 'Sweden']
```

```
In [91]: from scipy.cluster.hierarchy import linkage, dendrogram
         from scipy.spatial.distance import pdist
         from sklearn.preprocessing import StandardScaler
         from sklearn.neighbors import kneighbors_graph
         from sklearn.metrics import euclidean_distances
```

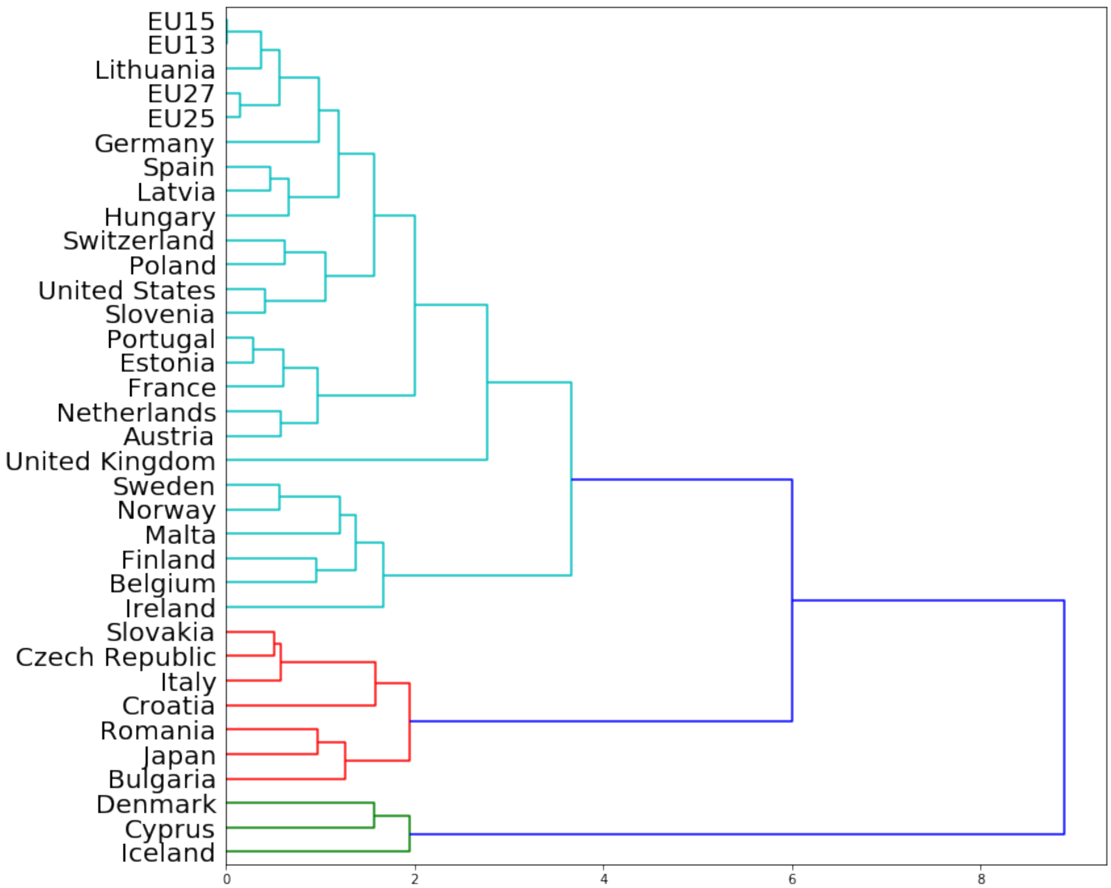
```
X = StandardScaler().fit_transform(edudrop.values)

distances = euclidean_distances(edudrop.values)

spectral = cluster.SpectralClustering(n_clusters=4, affinity="nearest_neighbors")
spectral.fit(edudrop.values)

y_pred = spectral.labels_.astype(np.int)
```

```
In [92]: idx=y_pred.argsort()

plt.plot(np.arange(35),y_pred[idx],'ro')
wrk_countries_names = [countries[i] for i,item in enumerate(wrk_countries) if item ]

plt.xticks(np.arange(len(wrk_countries_names)),[wrk_countries_names[i]
for i in idx],rotation=90,horizontalalignment='left',fontsize=12)

plt.yticks([0,1,2,3])

plt.title('Applying Spectral Clustering on the drop features',size=15)
fig = plt.gcf()
fig.set_size_inches((12,5))
```



Applying the agglomerative clustering, we obtain not only the different clusters, but also we can see how different clusters are obtained. This, in some way it is giving us information on which are the pairs of countries and clusters that are most similar. The corresponding code that applies the agglomerative clustering is:

```
In [93]: X_train = edudrop.values
         dist = pdist(X_train,'euclidean')
         linkage_matrix = linkage(dist,method = 'complete');
         plt.figure() # we need a tall figure
         fig = plt.gcf()
         fig.set_size_inches((12,12))
         dendrogram(linkage_matrix, orientation="right", color_threshold = 4,labels = wrk_countr

         plt.savefig("files/ch07/ACCountires.png",dpi=300, bbox_inches='tight')
         plt.show()

         #plt.tight_layout() # fixes margins
```



In scikit-learn, the parameter color_threshold colors all the descendent links below a cluster node k the same color if k is the first node below the color threshold. All links connecting nodes with distances greater than or equal to the threshold are colored blue. Thus, if we use color threshold = 3, the obtained clusters are as follows:

- Cluster 0: ['Cyprus', 'Denmark', 'Iceland']
- Cluster 1: ['Bulgaria', 'Croatia', 'Czech Republic', 'Italy', 'Japan', 'Romania', 'Slovakia']

- Cluster 2: ['Belgium', 'Finland', 'Ireland', 'Malta', 'Norway', 'Sweden']
- Cluster 3: ['Austria', 'Estonia', 'EU13', 'EU15', 'EU25', 'EU27', 'France', 'Germany', 'Hungary', 'Latvia', 'Lithuania', 'Netherlands', 'Poland', 'Portugal', 'Slovenia', 'Spain', 'Switzerland', 'United Kingdom', 'United States']

Note that they correspond in high degree to the clusters obtained by the K-means (except permutation of clusters labels that is irrelevant). The figure shows the construction of the clusters using the complete linkage agglomerative clustering. Different cuts at different levels of the dendrogram allow to obtain different number of clusters. As a summary, let us compare the results of the three approaches of clustering. We cannot expect that the results coincide since different approaches are based on different criteria to construct the clusters. Still, we can observe that in this case K-means and the agglomerative approaches gave the same results (up to a permutation of the number of cluster that is irrelevant), meanwhile the spectral clustering gave more evenly distributed clusters. It fused cluster 0 and 2 of the agglomerative clustering in cluster 1, and split cluster 3 of agglomerative clustering in clusters 0 and 3 of it. Note that these results can change when using different distance between data