Introduction-SciKit-Learn-Clustering

December 25, 2021

1 Introduction to Sci-Kit Learn and Clustering

In this tutorial we will introduce the Sci-Kit Learn library:https://scikit-learn.org/stable/

This is a very important library with a huge toolkit for data processing, unsupervised and supervised learning. It is one of the core tools for data science.

We will see some of the capabilities of this toolkit and focus on clustering.

[26]:
```
import numpy as np
import scipy as sp
import scipy.sparse as sp_sparse
import scipy.spatial.distance as sp_dist

import matplotlib.pyplot as plt

import sklearn as sk
import sklearn.datasets as sk_data
import sklearn.metrics as metrics
from sklearn import preprocessing
import sklearn.cluster as sk_cluster
import sklearn.feature_extraction.text as sk_text


import scipy.cluster.hierarchy as hr

import time
import seaborn as sns

%matplotlib inline
```

1.1 Computing distances

For the computation of distances there are libraries in Scipy

http://docs.scipy.org/doc/scipy-0.15.1/reference/spatial.distance.html#module-scipy.spatial.distance

but also in SciKit metrics library:

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise_distances.html

Most of these work with sparse data as well.

1.1.1 Compute distances using scipy

Computing distances between vectors

[28]:
```
import scipy.spatial.distance as sp_dist

x = np.random.randint(2, size = 5)
y = np.random.randint(2, size = 5)
print (x)
print (y)
print (sp_dist.cosine(x,y))
print (sp_dist.euclidean(x,y))
print (sp_dist.jaccard(x,y))
print (sp_dist.hamming(x,y))
# When computing jaccard similarity of 0/1 matrices,
# 1 means that the element corresponding to the column is in the set,
# 0 that the element is not in the set
```

```
[0 1 1 1 1]
[0 1 1 0 1]
0.1339745962155614
1.0
0.25
0.2
```

Compute pairwise distances in a table using **pdist** of scipy.

When given a matrix, it computes all pairwise distances between its rows. The output is a vector with N(N-1)/2 entries (N number of rows). We can transform it into an NxN distance matrix using **squareform**.

[30]:
```
A = np.random.randint(2, size = (5,3))

# computes the matrix of all pairwise distances of rows
# returns a vector with N(N-1)/2 entries (N number of rows)
D = sp_dist.pdist(A, 'jaccard')
print (A)
print('\n all row distances')
print (D)
print(sp_dist.squareform(D))
```

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

 all row distances
```

```
[1.         0.66666667 0.5        0.5        0.66666667 0.5
 1.         0.33333333 0.33333333 0.66666667]
[[0.         1.         0.66666667 0.5        0.5        ]
 [1.         0.         0.66666667 0.5        1.         ]
 [0.66666667 0.66666667 0.         0.33333333 0.33333333]
 [0.5        0.5        0.33333333 0.         0.66666667]
 [0.5        1.         0.33333333 0.66666667 0.        ]]
```

We can compute all pairwise distances between the rows of two tables A and B, using the **cdist** function of scipy. If A has N rows and B has M rows the result is an NxM matrix with all the distances

[33]:
```
B = np.random.randint(2, size = (3,3))
print(B)
D = sp_dist.cdist(A,B,'jaccard')
print(D)
```

```
[[1 1 1]
 [1 0 1]
 [1 1 0]]
[[0.66666667 0.5        0.5        ]
 [0.66666667 0.5        1.         ]
 [0.         0.33333333 0.33333333]
 [0.33333333 0.         0.66666667]
 [0.33333333 0.66666667 0.        ]]
```

1.1.2 Compute distances using sklearn

[34]:
```
import sklearn.metrics as metrics

#computes the matrix of all pairwise distances of rows
# returns a NxN matrix (N number of rows)
print(A)
D2 = metrics.pairwise_distances(A,metric = 'jaccard')
print('\n the matrix of row distances')
print(D2)
```

```
[[1 0 0]
 [0 0 1]
 [1 1 1]
 [1 0 1]
 [1 1 0]]
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\pairwise.py:1765:
DataConversionWarning: Data was converted to boolean for metric jaccard
  warnings.warn(msg, DataConversionWarning)


 the matrix of row distances
[[0.         1.         0.66666667 0.5        0.5        ]
```

```
 [1.         0.         0.66666667 0.5        1.        ]
 [0.66666667 0.66666667 0.         0.33333333 0.33333333]
 [0.5        0.5        0.33333333 0.         0.66666667]
 [0.5        1.         0.33333333 0.66666667 0.        ]]
```

Some similarity and distance metrics are directly computed in the pairwise library:

https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics.pairwise

[35]:
```
C = metrics.pairwise.cosine_similarity(A)
print('Cosine Similarity')
print(C)
```

```
Cosine Similarity
[[1.         0.         0.57735027 0.70710678 0.70710678]
 [0.         1.         0.57735027 0.70710678 0.        ]
 [0.57735027 0.57735027 1.         0.81649658 0.81649658]
 [0.70710678 0.70710678 0.81649658 1.         0.5       ]
 [0.70710678 0.         0.81649658 0.5        1.        ]]
```

Compute distances between the rows of two tables

[36]:
```
print(A)
print (B)

#computes the matrix of all pairwise distances of rows of A with rows of B
# returns an NxM matrix (N rows of A, M rows of B)
D3 = metrics.pairwise_distances(A,B,metric = 'jaccard')
print('\n the matrix of distances between the rows of A and B')
print(D3)
```

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

 the matrix of distances between the rows of A and B
[[0.66666667 0.5        0.5       ]
 [0.66666667 0.5        1.        ]
 [0.         0.33333333 0.33333333]
 [0.33333333 0.         0.66666667]
 [0.33333333 0.66666667 0.        ]]
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\pairwise.py:1765:
DataConversionWarning: Data was converted to boolean for metric jaccard
  warnings.warn(msg, DataConversionWarning)
```

We can apply everything to sparce matrices

[37]:
```
d = np.array([[0, 0, 12],
              [0, 1, 1],
              [0, 5, 34],
              [1, 3, 12],
              [1, 2, 6],
              [2, 0, 23],
              [3, 4, 14],
              ])
s = sp_sparse.csr_matrix((d[:,2],(d[:,0],d[:,1])), shape=(4,6))
D4 = metrics.pairwise.pairwise_distances(s,metric = 'euclidean')
print(s.toarray())
print(D4)
```

```
[[12  1  0  0  0 34]
 [ 0  0  6 12  0  0]
 [23  0  0  0  0  0]
 [ 0  0  0  0 14  0]]
[[ 0.         38.48376281 35.74912586 38.69108424]
 [38.48376281  0.         26.62705391 19.39071943]
 [35.74912586 26.62705391  0.         26.92582404]
 [38.69108424 19.39071943 26.92582404  0.        ]]
```

1.2 Clustering

You can read more about clustering in SciKit here:

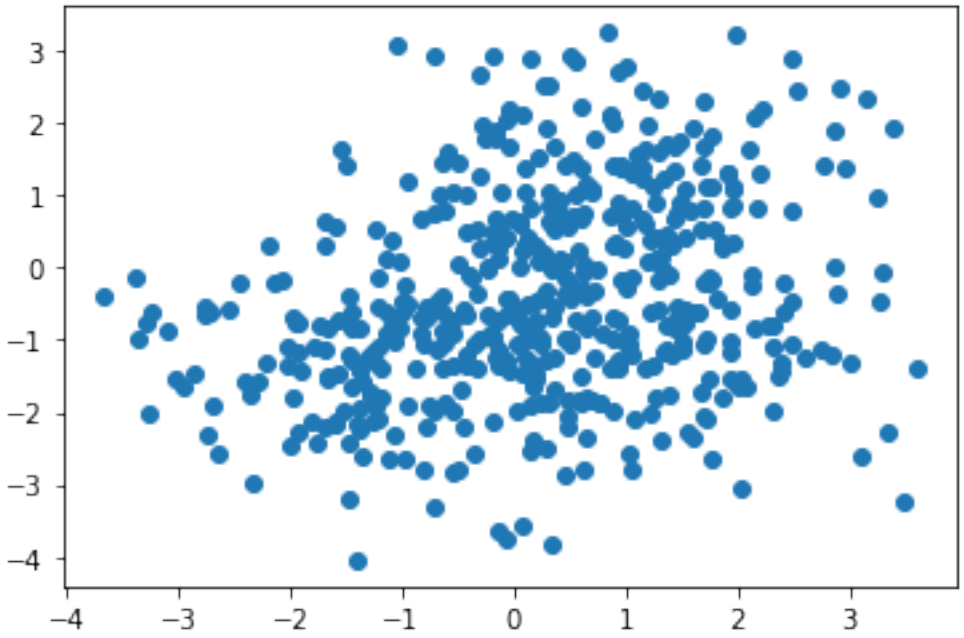
http://scikit-learn.org/stable/modules/clustering.html

Generate data from Gaussian distributions.

More on data generation here: http://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_blobs.htm

[39]:
```
centers = [[1,1], [-1, -1], [1, -1]]
X, true_labels = sk_data.make_blobs(n_samples=500, centers=centers,␣
 ↪n_features=2,
                                    center_box=(-10.0, 10.0),random_state=0)
plt.scatter(X[:,0], X[:,1])
```

[39]:
```
<matplotlib.collections.PathCollection at 0x26ba1eaa4c0>
```

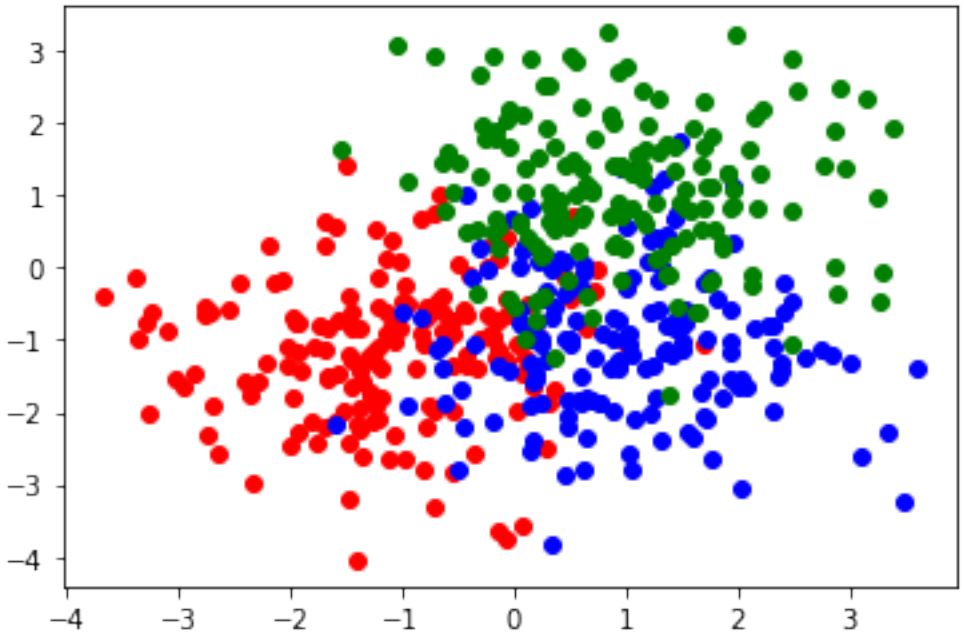


[40]:
```
print(type(X))
print(true_labels)
print(len(true_labels[true_labels==0]),len(true_labels[true_labels==1]),len(true_labels[true_l
```

```
<class 'numpy.ndarray'>
[2 0 1 1 0 1 1 2 2 2 1 0 0 1 1 0 0 2 2 0 0 0 2 2 0 1 0 2 0 2 0 0 0 2 1 1 0
 0 2 2 0 2 1 0 2 2 0 0 1 2 2 0 0 1 0 2 1 1 1 2 2 1 0 0 2 1 1 2 2 2 2 1 2 0
 0 0 2 2 0 0 0 0 0 2 1 2 2 0 0 2 2 1 0 2 1 0 1 2 1 1 2 2 1 2 1 0 1 1 0 2 2
 2 0 2 0 2 2 0 1 1 0 1 2 1 1 2 2 1 2 0 0 0 1 2 2 0 2 0 2 1 2 1 0 0 1 0 2 1
 0 1 2 2 2 0 1 0 1 0 2 2 0 1 0 0 1 2 1 1 1 2 1 2 1 0 1 0 2 2 0 2 1 0 2 2 0
 1 2 0 0 2 1 2 2 2 0 2 2 1 2 1 0 2 1 2 1 2 0 0 0 1 2 0 0 2 1 1 2 2 0 1 2 0
 0 1 1 1 0 2 2 2 1 2 1 1 1 0 1 2 0 2 1 2 0 2 1 1 2 2 1 2 0 0 1 0 1 0 2 1 2
 1 1 1 1 0 0 1 0 1 1 1 1 2 1 0 0 0 0 2 1 2 2 0 0 1 0 2 1 0 2 2 1 0 0 1 0 2
 1 2 1 0 0 0 1 2 0 0 2 2 1 0 0 1 1 0 2 1 0 1 2 1 1 0 2 0 2 1 2 1 0 0 0 1 0
 2 2 1 0 2 2 2 0 1 1 1 0 1 0 0 0 2 0 2 0 2 2 0 2 2 2 2 1 1 1 2 2 2 2 0 0 0
 1 2 0 1 0 1 0 1 2 2 0 2 1 0 1 2 2 0 1 2 1 2 0 0 0 1 2 0 0 1 2 2 0 2 1 0 2
 0 1 0 2 0 0 1 0 0 0 0 1 0 2 2 2 1 0 1 2 1 2 1 0 1 1 1 1 1 0 2 1 2 0 0 1 2
 0 2 1 0 0 1 1 2 1 2 1 1 1 1 2 0 1 1 0 1 2 0 1 1 0 2 0 0 1 1 1 0 1 0 1 1 1
 1 0 1 1 2 2 2 1 1 1 0 1 2 2 2 1 0 0 2]
167 167 166
```

[41]:
```
plt.scatter(X[true_labels==1,0], X[true_labels==1,1],c = 'r')
plt.scatter(X[true_labels==2,0], X[true_labels==2,1],c = 'b')
plt.scatter(X[true_labels==0,0], X[true_labels==0,1],c = 'g')
```

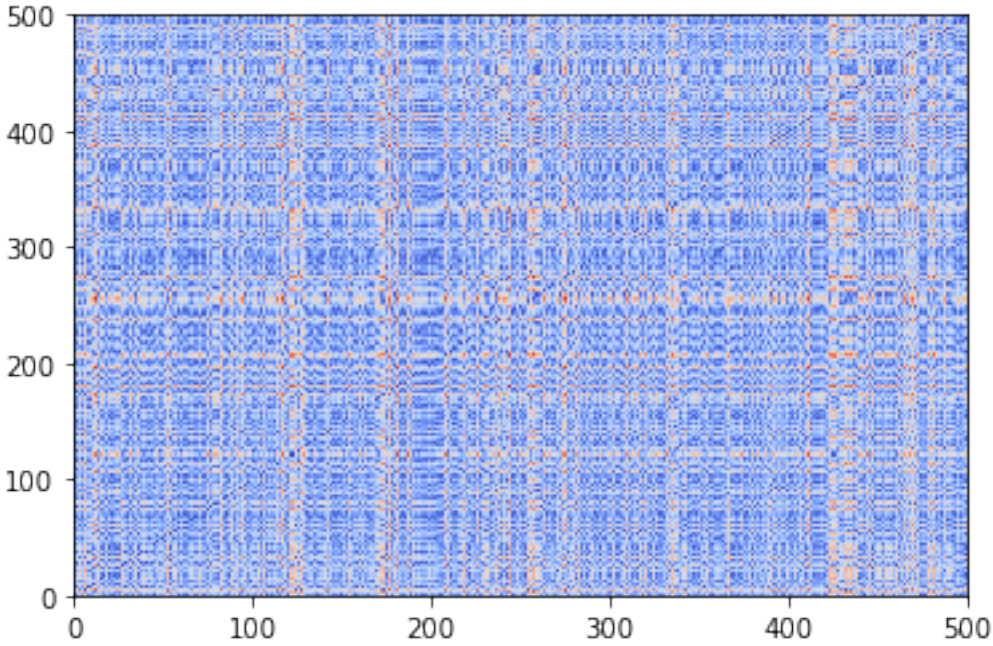
[41]: <matplotlib.collections.PathCollection at 0x26ba1f09430>



Useful command: We will create a colormap of the distance matrix using the pcolormesh method of matplotlib.pyplot

[42]:
```
euclidean_dists = metrics.euclidean_distances(X)
plt.pcolormesh(euclidean_dists,cmap=plt.cm.coolwarm)
```

[42]: <matplotlib.collections.QuadMesh at 0x26ba1f63bb0>



1.3 Clustering Algorithms

scikit-learn has a huge set of tools for unsupervised learning generally, and clustering specifically. These are in sklearn.cluster. http://scikit-learn.org/stable/modules/clustering.html

There are 3 functions in all the clustering classes,

fit(): builds the model from the training data (e.g. for kmeans, it finds the centroids)

predict(): assigns labels to the data after building the model

fit_predict(): does both at the same data (e.g in kmeans, it finds the centroids and assigns the labels to the dataset)

1.3.1 K-means clustering

More on the k-means clustering here: http://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.ht

Important parameters

init: determines the way the initialization is done. kmeans++ is the default.

n_init: number of iterations

Important attributes:

labels_ the labels for each point

cluster_centers_: the cluster centroids

inertia_: the SSE value

[43]:
```
import sklearn.cluster as sk_cluster

kmeans = sk_cluster.KMeans(init='k-means++', n_clusters=3, n_init=10)
kmeans.fit_predict(X)
centroids = kmeans.cluster_centers_
kmeans_labels = kmeans.labels_
error = kmeans.inertia_

print ("The total error of the clustering is: ", error)
print ('\nCluster labels')
print(kmeans_labels)
print ('\n Cluster Centroids')
print (centroids)
```

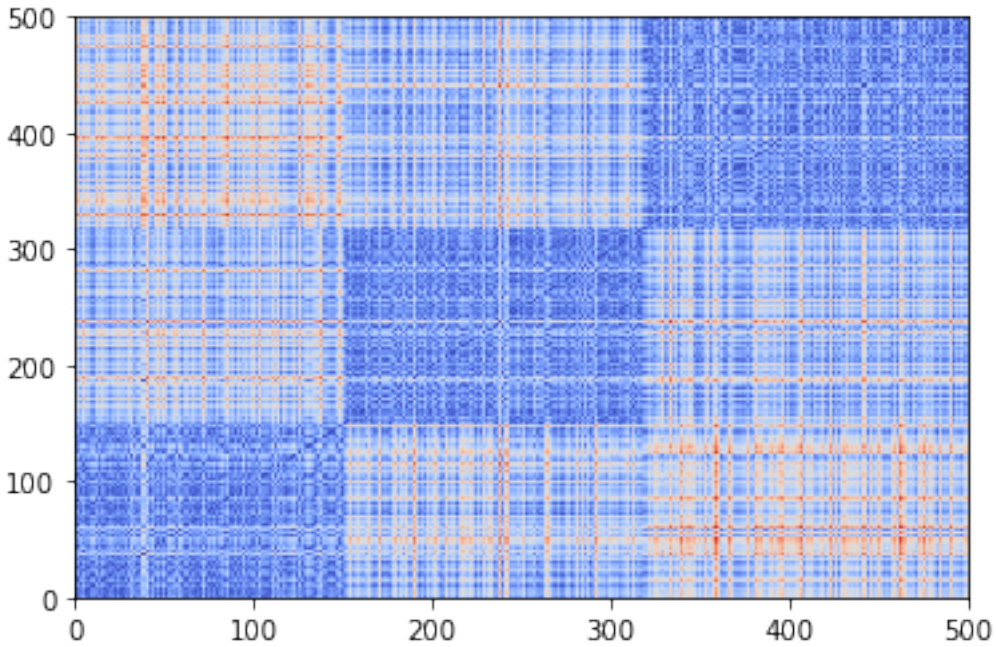
```
The total error of the clustering is:  729.3882206069684

Cluster labels
[1 1 1 0 2 0 0 1 1 1 0 2 2 0 0 2 2 2 0 1 2 2 1 2 2 0 2 0 2 1 1 2 2 1 0 0 2
 2 1 1 2 1 0 2 1 1 2 2 1 1 0 2 2 0 2 2 0 0 2 2 1 0 2 2 2 0 0 0 1 1 2 0 1 2
```

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

 Cluster Centroids
[[-1.3362657  -1.28432839]
 [ 1.14789815 -1.17752675]
 [ 0.78589165  1.17781335]]
```

Useful command: numpy.argsort sorts a set of values and returns the sorted indices

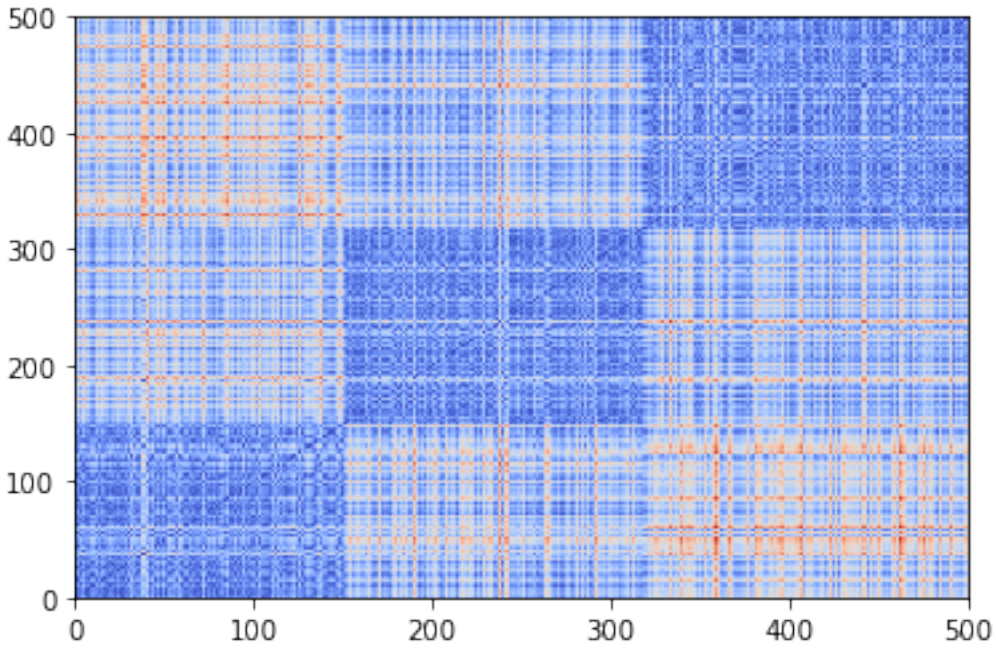
[44]:
```
idx = np.argsort(kmeans_labels) # returns the indices in sorted order
rX = X[idx,:]
r_euclid = metrics.euclidean_distances(rX)
#r_euclid = euclidean_dists[idx,:][:,idx]
plt.pcolormesh(r_euclid,cmap=plt.cm.coolwarm)
```

[44]:
```
<matplotlib.collections.QuadMesh at 0x26ba1fb6850>
```



[49]:
```
se = euclidean_dists[idx,:]
se = se[:,idx]
plt.pcolormesh(se,cmap=plt.cm.coolwarm)
```

[49]: <matplotlib.collections.QuadMesh at 0x26ba6c2e2e0>



Confusion matrix: http://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html

Important: In the produced confusion matrix, the first list defines the rows and the second the columns. The matrix is always square, regarless if the number of classes and clusters are not the same. The extra rows or columns are filled with zeros.

Homogeneity and completeness: http://scikit-learn.org/stable/modules/clustering.html#homogeneity-completeness

Homogeneity and completeness are computed using the conditional entropy of the labels given the cluster, and the conditional entropy of the cluster labels given the class label. The V-measure combines these in a similar way like F-measure

Precision: http://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_score.html#sklearn.metric

Recall: http://scikit-learn.org/stable/modules/generated/sklearn.metrics.recall_score.html#sklearn.metrics.recal

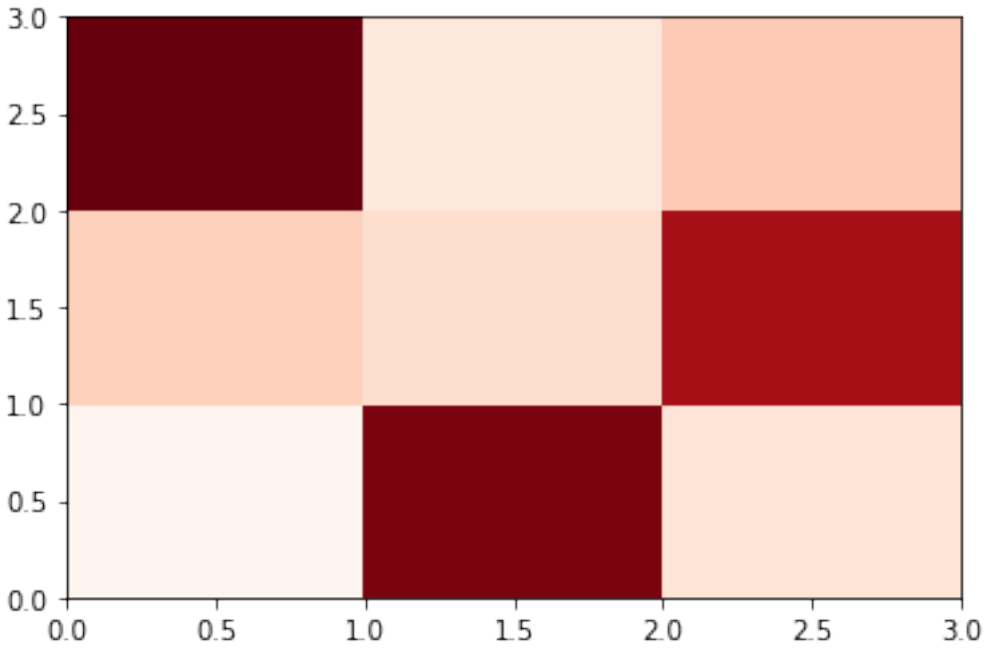
Silhouette score: http://scikit-learn.org/stable/modules/generated/sklearn.metrics.silhouette_score.html

[50]:
```
C= metrics.confusion_matrix(kmeans_labels,true_labels)
print (C)
```

```
plt.pcolormesh(C,cmap=plt.cm.Reds)
```

```
[[  1 135  15]
 [ 26  20 122]
 [140  12  29]]
```

[50]: <matplotlib.collections.QuadMesh at 0x26ba6c7ca90>



Compute precision and recall.

These metrics are for classification, so they assume that row i is mapped to column i

[51]:
```
p = metrics.precision_score(true_labels,kmeans_labels, average=None)
print(p)
r = metrics.recall_score(true_labels,kmeans_labels, average = None)
print(r)
```

```
[0.00662252 0.11904762 0.16022099]
[0.00598802 0.11976048 0.1746988 ]
```

Create a function that maps each cluster to the class that has the most points.

You need to be careful if many clusters map to the same class. It will not work in this case

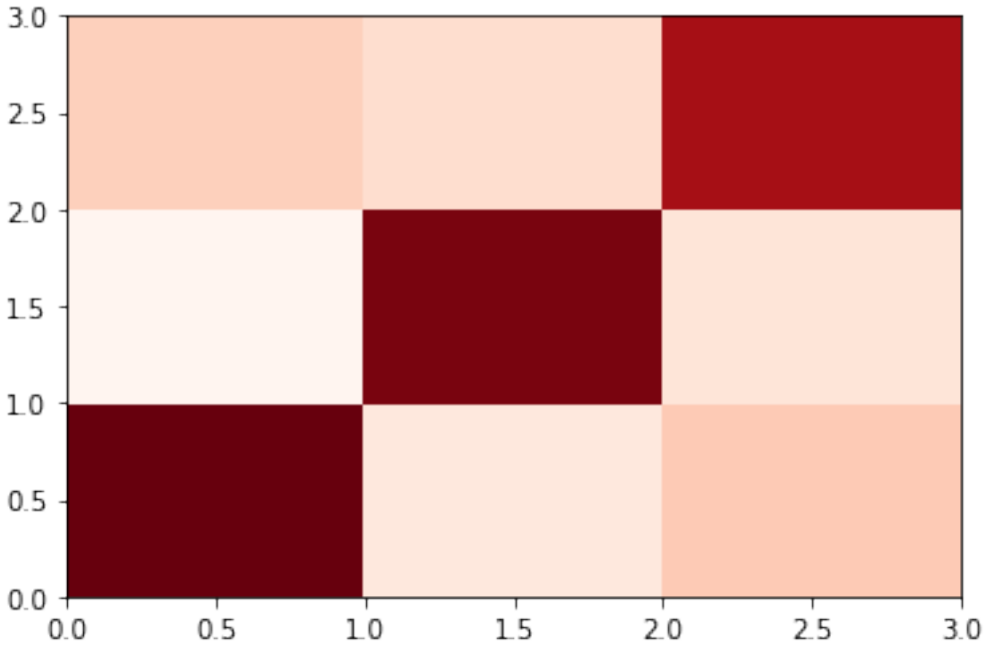
Useful command: numpy.argmax returns the index of the max element

[52]:
```
def cluster_class_mapping(kmeans_labels,true_labels):
    C= metrics.confusion_matrix(kmeans_labels,true_labels)
```

```
    mapping = list(np.argmax(C,axis=1)) #for each row (cluster) find the best␣
 ↪class in the confusion matrix
    mapped_kmeans_labels = [mapping[l] for l in kmeans_labels]
    C2= metrics.confusion_matrix(mapped_kmeans_labels,true_labels)
    return mapped_kmeans_labels,C2

mapped_kmeans_labels,C = cluster_class_mapping(kmeans_labels,true_labels)
print(C)
plt.pcolormesh(C, cmap=plt.cm.Reds)
```

```
[[140  12  29]
 [  1 135  15]
 [ 26  20 122]]
```

[52]: `<matplotlib.collections.QuadMesh at 0x26ba6d247c0>`



Compute different metrics for clustering quality

[53]:
```
h = metrics.homogeneity_score(true_labels,mapped_kmeans_labels)
print(h)
c = metrics.completeness_score(true_labels,mapped_kmeans_labels)
print(c)
v = metrics.v_measure_score(true_labels,mapped_kmeans_labels)
print(v)
p = metrics.precision_score(true_labels,mapped_kmeans_labels, average=None)
print(p)
```

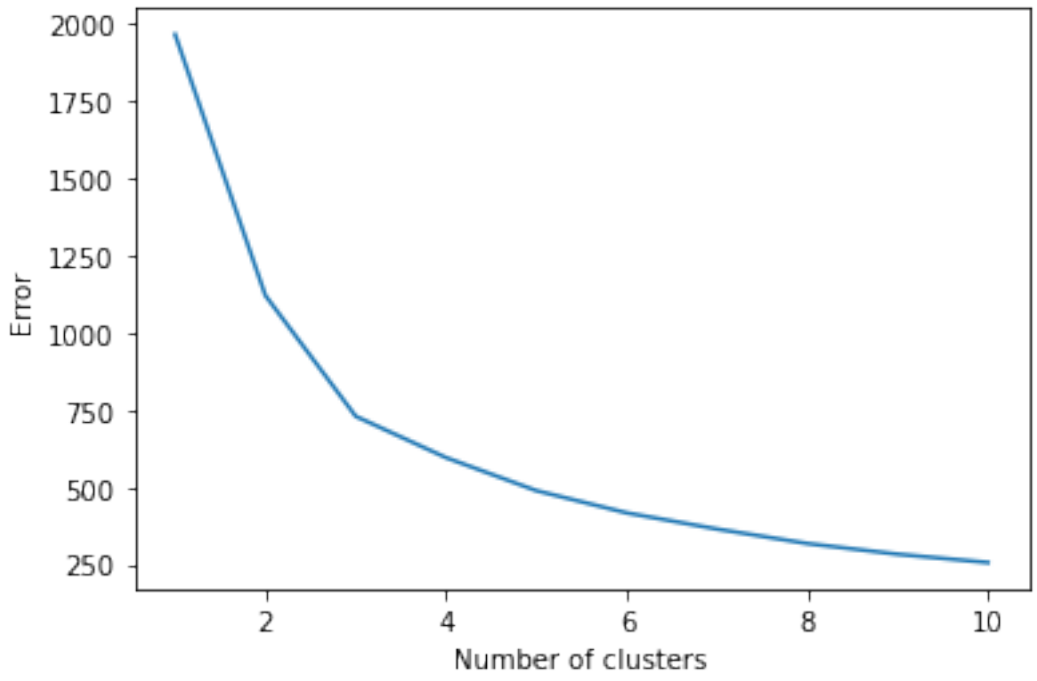
```
r = metrics.recall_score(true_labels,mapped_kmeans_labels, average = None)
print(r)
f = metrics.f1_score(true_labels,mapped_kmeans_labels, average = None)
print(f)
p = metrics.precision_score(true_labels,mapped_kmeans_labels,␣
 ↪average='weighted')
print(p)
r = metrics.recall_score(true_labels,mapped_kmeans_labels, average = 'weighted')
print(r)
f = metrics.f1_score(true_labels,mapped_kmeans_labels, average = 'weighted')
print(f)
```

```
0.44199547480098583
0.4430951461741084
0.44254462735008065
[0.77348066 0.89403974 0.72619048]
[0.83832335 0.80838323 0.73493976]
[0.8045977  0.8490566  0.73053892]
0.7980470510548809
0.794
0.794859459999974
```

The SSE plot

[54]:
```
error = np.zeros(11)
sh_score = np.zeros(11)
for k in range(1,11):
    kmeans = sk_cluster.KMeans(init='k-means++', n_clusters=k, n_init=10)
    kmeans.fit_predict(X)
    error[k] = kmeans.inertia_
    if k>1: sh_score[k]= metrics.silhouette_score(X, kmeans.labels_)

plt.plot(range(1,len(error)),error[1:])
plt.xlabel('Number of clusters')
plt.ylabel('Error')
```

[54]:
```
Text(0, 0.5, 'Error')
```

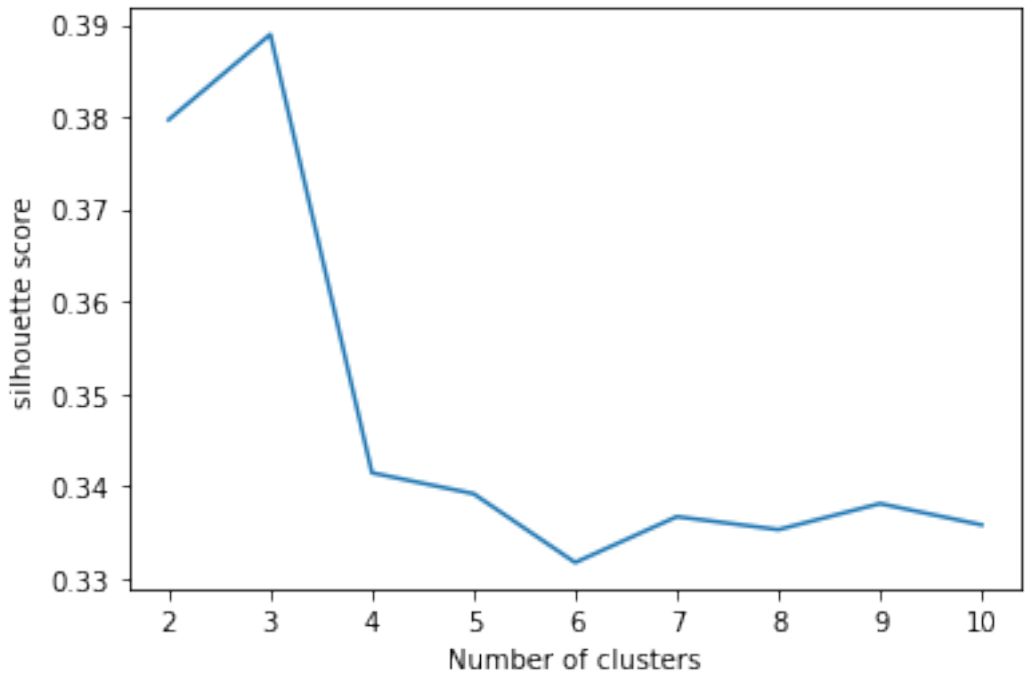


The silhouette plot

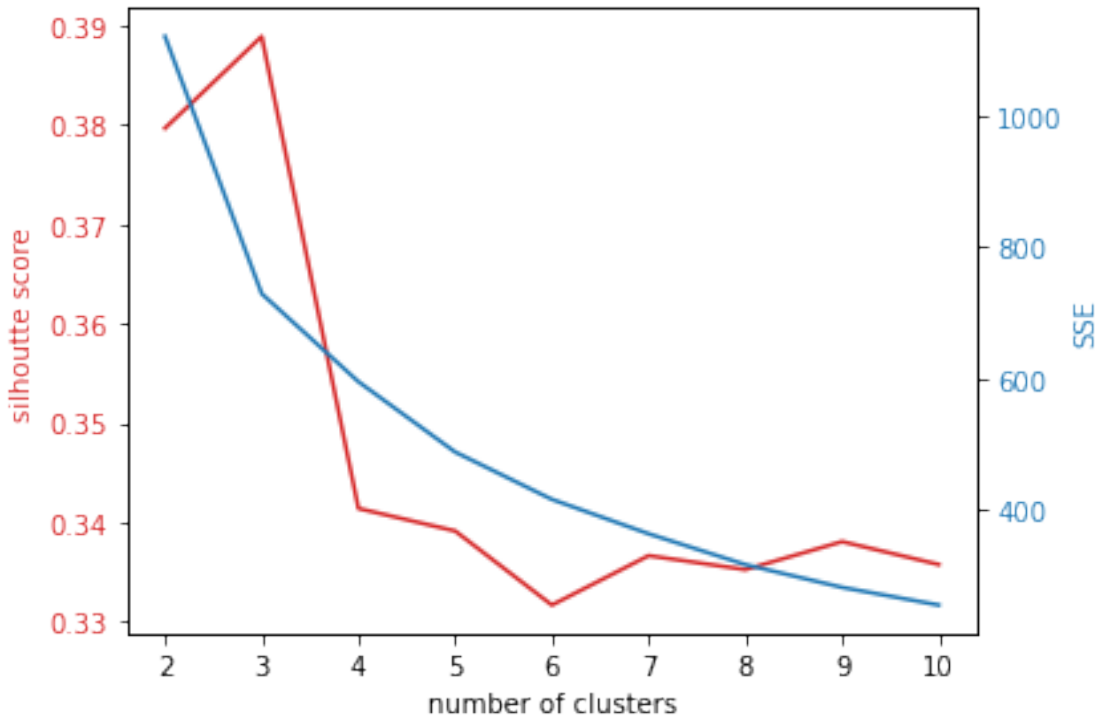
We see a peak at k = 3 and k = 6 indicating that these may be good values for the cluster number

[55]:
```
plt.plot(range(2,len(sh_score)),sh_score[2:])
plt.xlabel('Number of clusters')
plt.ylabel('silhouette score')
```

[55]:
```
Text(0, 0.5, 'silhouette score')
```

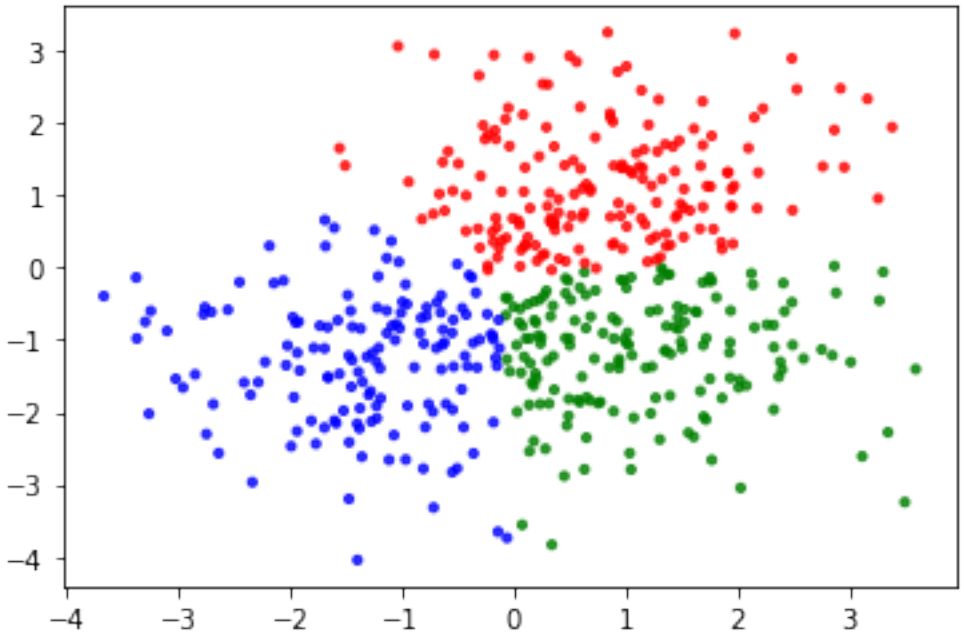


[56]:
```
fig, ax1 = plt.subplots()

color = 'tab:red'
ax1.set_xlabel('number of clusters')
ax1.set_ylabel('silhoutte score', color=color)
ax1.plot(range(2,len(sh_score)),sh_score[2:], color=color)
ax1.tick_params(axis='y', labelcolor=color)

ax2 = ax1.twinx()  # instantiate a second axes that shares the same x-axis

color = 'tab:blue'
ax2.set_ylabel('SSE', color=color)  # we already handled the x-label with ax1
ax2.plot(range(2,len(error)),error[2:], color=color)
ax2.tick_params(axis='y', labelcolor=color)

fig.tight_layout()
```



[57]:
```
colors = np.array([x for x in 'bgrcmykbgrcmykbgrcmykbgrcmyk'])
colors = np.hstack([colors] * 20)
plt.scatter(X[:, 0], X[:, 1], color=colors[kmeans_labels].tolist(), s=10,␣
↪alpha=0.8)
```

[57]:
```
<matplotlib.collections.PathCollection at 0x26ba71e40a0>
```



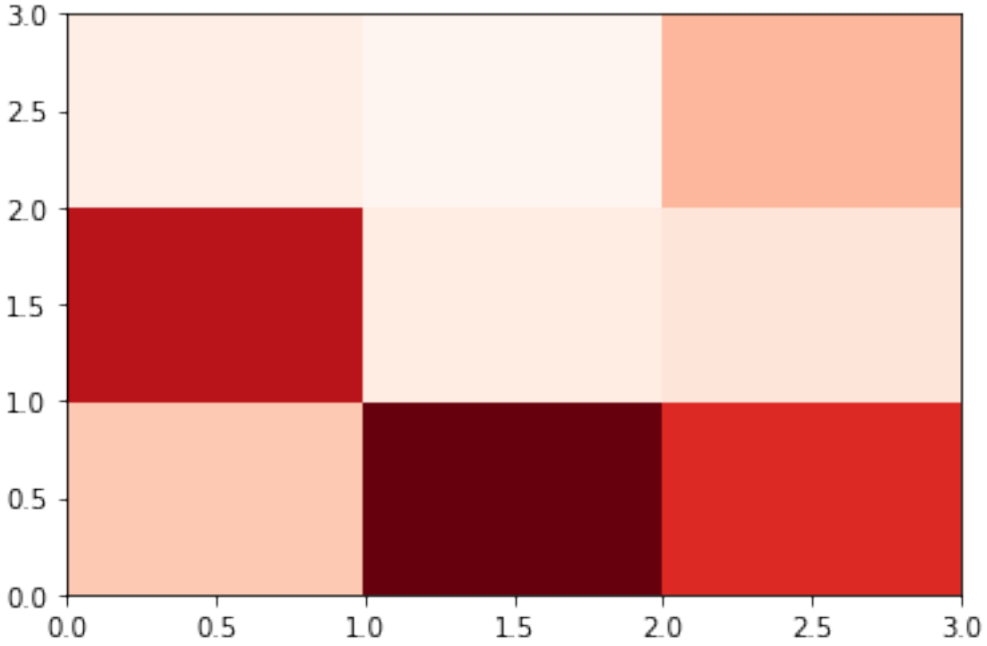
1.3.2 Agglomerative Clustering

More on Agglomerative Clustering here: http://scikit-learn.org/stable/modules/generated/sklearn.cluster.Agglome

[58]:
```
agglo = sk_cluster.AgglomerativeClustering(linkage = 'complete', n_clusters = 3)
agglo_labels = agglo.fit_predict(X)

C_agglo= metrics.confusion_matrix(agglo_labels,true_labels)
print (C_agglo)
#plt.pcolor(C_agglo,cmap=plt.cm.coolwarm)
plt.pcolormesh(C_agglo,cmap=plt.cm.Reds)

mapped_agglo_labels,C_agglo = cluster_class_mapping(agglo_labels,true_labels)
print(C_agglo)
p = metrics.precision_score(true_labels,mapped_agglo_labels, average='weighted')
print(p)
r = metrics.recall_score(true_labels,mapped_agglo_labels, average = 'weighted')
print(r)
```

```
[[ 33 156 108]
 [126  10  16]
 [  8   1  42]]
[[126  10  16]
 [ 33 156 108]
 [  8   1  42]]
0.7257145291928573
```

```
0.648
```



Another way to do agglomerative clustering using SciPy:

https://docs.scipy.org/doc/scipy/reference/cluster.hierarchy.html

[59]:
```
import scipy.cluster.hierarchy as hr

Z = hr.linkage(X, method='complete', metric='euclidean')

print (Z.shape, X.shape)
```

```
(499, 4) (500, 2)
```

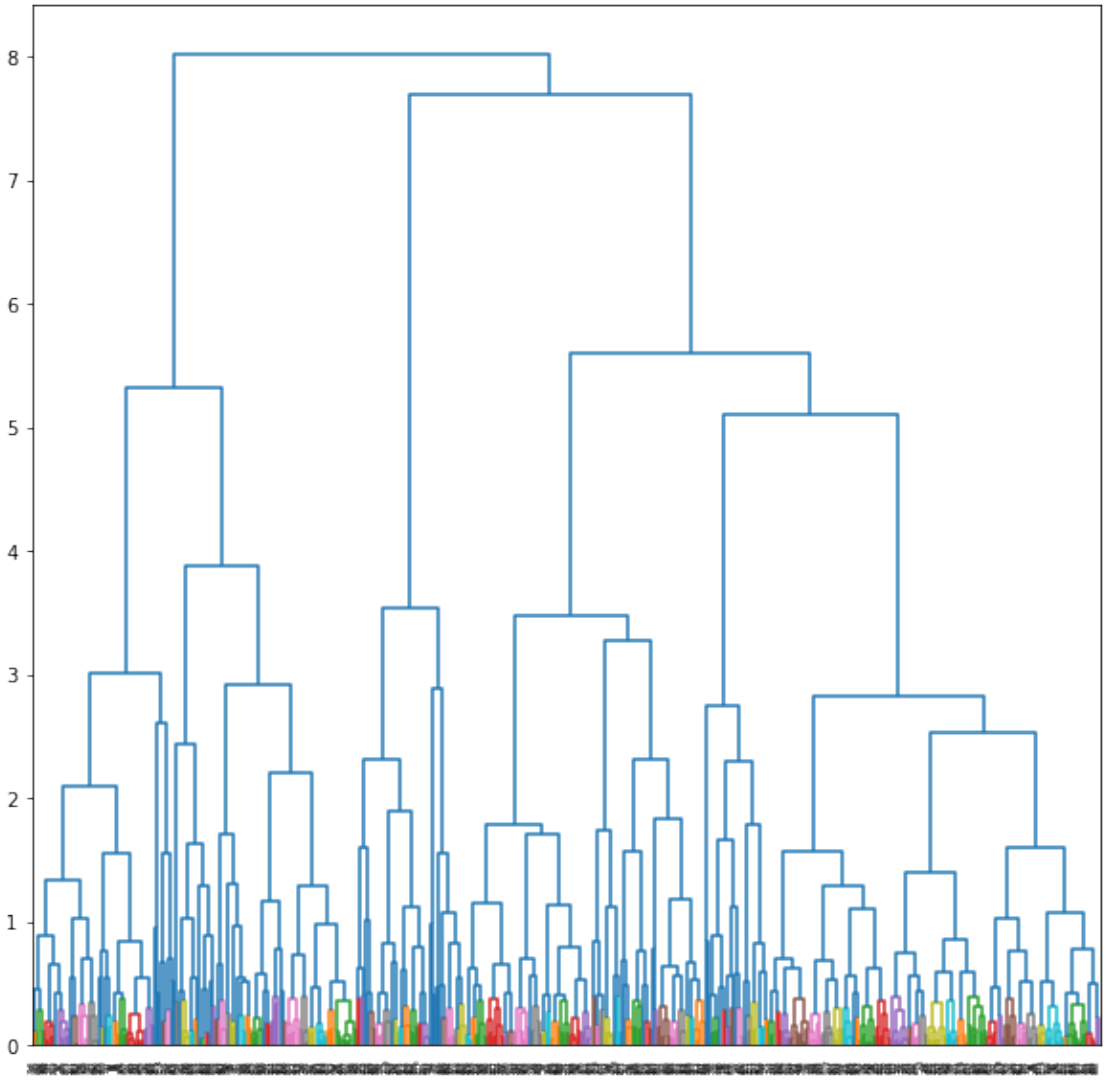
[60]:
```
import scipy.spatial.distance as sp_dist
D = sp_dist.pdist(X, 'euclidean')
Z = hr.linkage(D, method='complete')
print (Z.shape, X.shape)
```

```
(499, 4) (500, 2)
```

Hierarchical clustering returns a 4 by (n-1) matrix Z. At the i-th iteration, clusters with indices Z[i, 0] and Z[i, 1] are combined to form cluster n + i. A cluster with an index less than n corresponds to one of the n original observations. The distance between clusters Z[i, 0] and Z[i, 1] is given by Z[i, 2]. The fourth value Z[i, 3] represents the number of original observations in the newly formed cluster.

[61]:
```
fig = plt.figure(figsize=(10,10))
T = hr.dendrogram(Z,color_threshold=0.4, leaf_font_size=4)
fig.show()
```

```
<ipython-input-61-ddcad35f0acd>:3: UserWarning: Matplotlib is currently using
module://ipykernel.pylab.backend_inline, which is a non-GUI backend, so cannot
show the figure.
  fig.show()
```



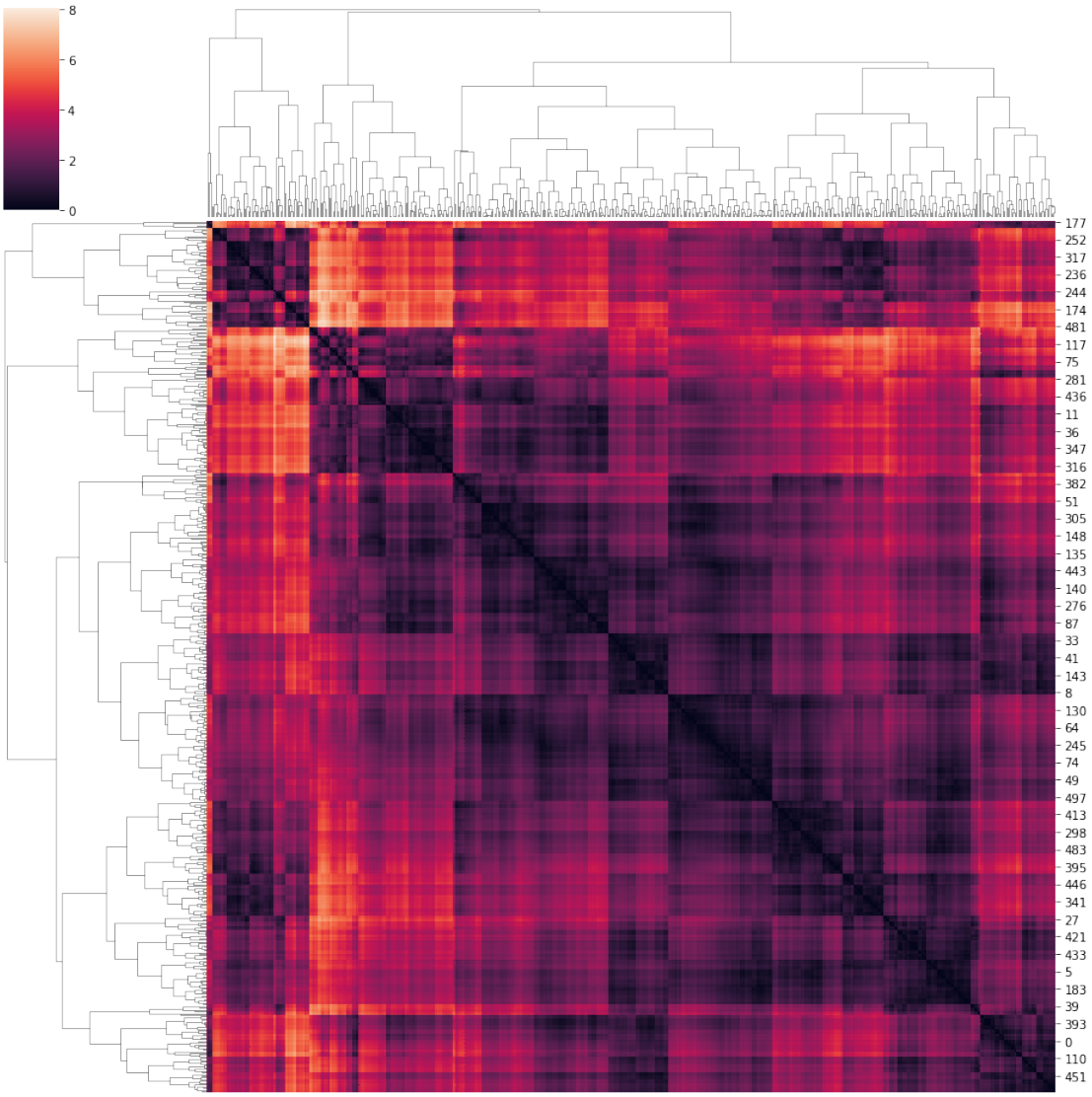
Another way to do agglomerative clustering (and visualizing it): http://seaborn.pydata.org/generated/seaborn.clustermap.html

[62]:
```
distances = metrics.euclidean_distances(X)
```

```
cg = sns.clustermap(distances, method="complete", figsize=(13,13),␣
 ↪xticklabels=False)
print (cg.dendrogram_col.reordered_ind)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\matrix.py:659: UserWarning:
Clustering large matrix with scipy. Installing `fastcluster` may give better
performance.
  warnings.warn(msg)
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\matrix.py:629:
ClusterWarning: scipy.cluster: The symmetric non-negative hollow observation
matrix looks suspiciously like an uncondensed distance matrix
  linkage = hierarchy.linkage(self.array, method=self.method,
```

```
[177, 469, 83, 179, 343, 61, 34, 124, 3, 466, 252, 490, 442, 312, 354, 230, 240,
476, 302, 57, 317, 154, 438, 167, 71, 472, 232, 399, 450, 35, 236, 454, 172,
478, 219, 107, 320, 455, 283, 434, 244, 426, 425, 121, 123, 25, 335, 432, 254,
6, 174, 127, 388, 423, 267, 53, 435, 257, 197, 209, 481, 415, 491, 264, 206,
294, 181, 411, 275, 12, 117, 208, 226, 187, 332, 444, 238, 274, 263, 310, 75,
355, 374, 4, 424, 465, 95, 114, 142, 309, 281, 129, 468, 471, 80, 250, 200, 266,
419, 235, 436, 383, 194, 462, 28, 160, 441, 301, 412, 40, 11, 173, 242, 380,
100, 350, 221, 330, 392, 410, 36, 499, 287, 394, 88, 31, 54, 43, 155, 182, 347,
20, 222, 295, 369, 32, 15, 188, 440, 326, 316, 447, 24, 82, 137, 92, 480, 168,
223, 431, 382, 484, 492, 52, 345, 363, 58, 300, 47, 78, 51, 387, 356, 145, 207,
346, 416, 62, 329, 37, 305, 98, 321, 153, 178, 247, 348, 306, 417, 112, 148,
163, 405, 367, 81, 255, 323, 304, 131, 313, 135, 218, 402, 120, 16, 482, 63,
205, 333, 474, 443, 231, 1, 403, 150, 108, 397, 45, 19, 376, 140, 357, 23, 282,
189, 398, 237, 239, 175, 212, 276, 284, 420, 372, 414, 26, 368, 318, 46, 299,
87, 211, 273, 430, 253, 17, 55, 477, 196, 319, 33, 279, 385, 277, 427, 364, 192,
195, 307, 38, 41, 29, 258, 136, 453, 115, 220, 458, 365, 400, 143, 157, 111,
475, 289, 216, 422, 97, 324, 159, 8, 353, 86, 265, 158, 225, 409, 204, 149, 213,
130, 340, 371, 79, 214, 233, 377, 73, 292, 486, 64, 269, 21, 328, 70, 105, 184,
228, 59, 493, 245, 251, 327, 7, 291, 344, 485, 103, 496, 370, 74, 401, 165, 249,
384, 94, 375, 147, 448, 30, 49, 198, 69, 201, 68, 459, 186, 134, 418, 243, 497,
72, 379, 185, 215, 13, 99, 351, 268, 373, 413, 90, 479, 106, 360, 463, 144, 166,
10, 56, 298, 362, 224, 234, 65, 488, 42, 202, 156, 50, 483, 311, 437, 278, 404,
118, 489, 325, 66, 457, 395, 270, 429, 91, 260, 67, 248, 14, 456, 358, 446, 296,
164, 96, 452, 199, 272, 315, 104, 308, 341, 261, 290, 141, 259, 342, 467, 180,
390, 190, 27, 460, 349, 210, 361, 76, 378, 109, 60, 116, 421, 193, 227, 138,
133, 101, 126, 44, 122, 151, 433, 2, 48, 113, 84, 408, 461, 288, 18, 473, 5,
119, 293, 132, 359, 176, 139, 286, 331, 449, 183, 336, 170, 191, 161, 381, 77,
262, 246, 439, 39, 171, 256, 280, 89, 366, 352, 470, 93, 322, 393, 498, 297,
396, 285, 203, 391, 314, 406, 464, 0, 407, 338, 152, 22, 271, 303, 128, 241,
487, 110, 162, 217, 229, 389, 494, 339, 146, 337, 102, 451, 386, 445, 9, 428,
169, 125, 495, 85, 334]
```



1.3.3 DBSCAN Algorithm

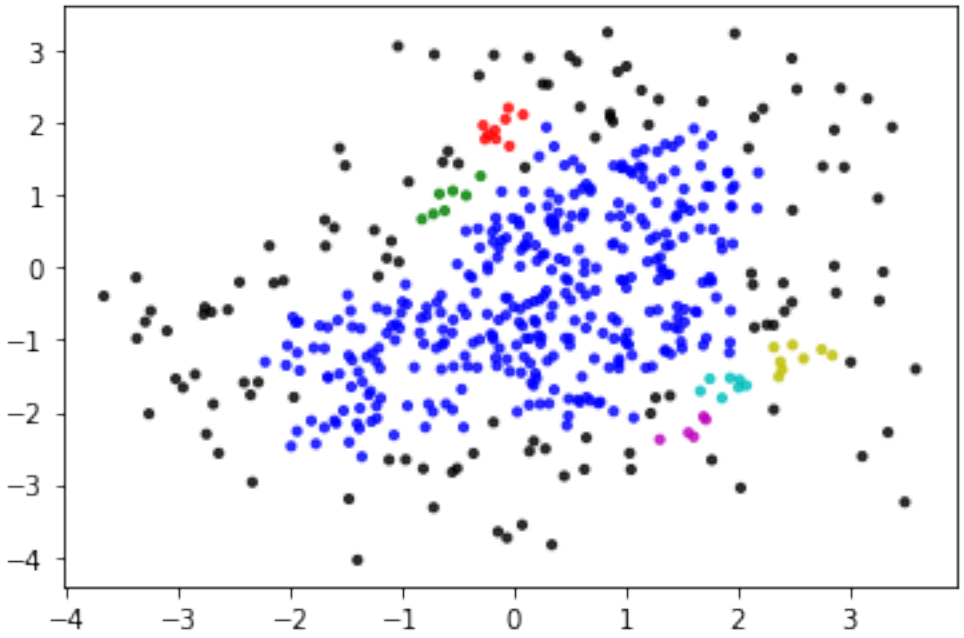
More on DBSCAN here: http://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html

[63]:
```
dbscan = sk_cluster.DBSCAN(eps=0.3)
dbscan_labels = dbscan.fit_predict(X)
print(dbscan_labels) #label -1 corresponds to noise
renamed_dbscan_labels = [x+1 for x in dbscan_labels]
C = metrics.confusion_matrix(renamed_dbscan_labels,true_labels)
#print(C)
print (C[:,:max(true_labels)+1])
```

```
[ 5  0  0 -1 -1  0 -1  0  0  3  0  0 -1 -1  0 -1  0  0  0  0  0  0  5  0
  0 -1  0 -1 -1  0  0  0 -1  0 -1  0  0  0  0 -1  0  0  0  0  0  0  0 -1
  0  0  0 -1  1 -1  0  0  0  0  1  0  0 -1  0  0  0  0  0  0  0  0  0 -1
  0  0  0 -1  0  0 -1  0  2  0  0 -1  0  3  0  0  0 -1  0  0 -1  0  0 -1
  0  0  0 -1  0  0  0  0  0  0  0  0  0  0 -1  0  0  0 -1  0  0 -1  0  0
  0 -1  0 -1 -1  3  0 -1 -1 -1  0  0  0  0  0  0  0  0  0  0  0  0 -1  0
  0  0  4  0  0  0  0  0  5  0  0  0  0  0  0  0  0  0  0 -1  1  0  0  0 -1
 -1  3  0 -1 -1  0 -1  0  0 -1  0 -1  0 -1  0  0  0  0  0 -1 -1  0 -1  0
  0  0  0  0  0 -1  0  0  2  0  0 -1  0  0 -1  0 -1 -1 -1  0  0  0  0 -1
  0 -1  0  0  0 -1  0 -1  0  0 -1  0  0  0  0  0  0  0  0  2  0  0 -1  0
  0  5  0  0 -1  0  0  0  0  0  2  0 -1  0 -1  0 -1 -1  0  0  0  0  0 -1
 -1  0  2 -1  0  0  0  5  0  0 -1 -1  0  0  0  0 -1 -1  0  0  0 -1  0  0
  0  0  0  0  0  0 -1  0  0  0  0  0  1  0  0 -1  0  0  0  0  0 -1 -1  0
  0  0 -1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 -1  0 -1  0  3 -1
  0  4  5  4  0  0  0 -1  0  1  0  0  0 -1  0  0 -1  0  0 -1  0  0  0  0
  0 -1  0  1  0  0 -1  0  0  0  0  0  0  0 -1  0  0  0 -1  0  0  0 -1  2
  0  0  0 -1 -1  4  0 -1  0  0  0  0  0  0  0  0  0  0  0  0  0 -1 -1  5
  0  0  0 -1  0  0  0 -1  0  0  0  2  0 -1  0 -1 -1 -1 -1  0  3  0  0 -1
 -1  0  0 -1  2  0  0  0 -1  0  0  0 -1  0  0  0  0  0  0  0  0  0 -1 -1
  0  0  0  0 -1  0  0  0 -1 -1 -1  0 -1 -1 -1  2 -1  0  0  0  0  0 -1  0
 -1 -1  0  0 -1  0  0  5  0  0 -1 -1  1  0  4  3  0  0  0  0]
[[ 48  47  26]
 [106 117 120]
 [  3   3   1]
 [  9   0   0]
 [  0   0   7]
 [  0   0   5]
 [  1   0   7]]
```

[64]:
```
#colors = np.array([x for x in 'bgrcmykbgrcmykbgrcmykbgrcmyk'])
#colors = np.hstack([colors] * 20)
colors = np.array([x for x in 'bgrcmywk'*10])
plt.scatter(X[:, 0], X[:, 1], color=colors[dbscan_labels].tolist(), s=10,␣
↪alpha=0.8)
```

[64]:
```
<matplotlib.collections.PathCollection at 0x26ba86f88b0>
```



1.4 Clustering text data

An example of what we want to do: http://scikit-learn.org/stable/auto_examples/text/document_clustering.html

SciKit datasets: http://scikit-learn.org/stable/datasets/

We will use the 20-newsgroups datasets which consists of postings on 20 different newsgroups.

More information here: http://scikit-learn.org/stable/datasets/#the-20-newsgroups-text-dataset

[65]:
```
from sklearn.datasets import fetch_20newsgroups

categories = ['comp.os.ms-windows.misc', 'sci.space','rec.sport.baseball']
#categories = ['alt.atheism', 'sci.space','rec.sport.baseball']
news_data = sk_data.fetch_20newsgroups(subset='train',
                                       remove=('headers', 'footers', 'quotes'),
                                       categories=categories)
print (news_data.target)
print (len(news_data.target))
```

```
[2 0 0 … 2 1 2]
1781
```

[66]:
```
print (type(news_data))
print (news_data.filenames)
print (news_data.target[:10])
print (news_data.data[1])
print (len(news_data.data))
```

```
<class 'sklearn.utils.Bunch'>
['C:\\Users\\tsap\\scikit_learn_data\\20news_home\\20news-bydate-
train\\sci.space\\60940'
 'C:\\Users\\tsap\\scikit_learn_data\\20news_home\\20news-bydate-
train\\comp.os.ms-windows.misc\\9955'
 'C:\\Users\\tsap\\scikit_learn_data\\20news_home\\20news-bydate-
train\\comp.os.ms-windows.misc\\9846'
 …
 'C:\\Users\\tsap\\scikit_learn_data\\20news_home\\20news-bydate-
train\\sci.space\\60891'
 'C:\\Users\\tsap\\scikit_learn_data\\20news_home\\20news-bydate-
train\\rec.sport.baseball\\104484'
 'C:\\Users\\tsap\\scikit_learn_data\\20news_home\\20news-bydate-
train\\sci.space\\61110']
[2 0 0 2 0 0 1 2 2 1]
Recently the following problem has arrisen.  The first time I turn on my
computer when windows starts (from my autoexec) after the win31 title screen
the computer reboots on its own.  Usually the second time (after reboot) or
from the DOS prompt everything works fine.

 s far as I remember I have not changed my config.sys or autoxec.bat or
win.ini.  I can't remember whether this problem occured before I
optimized/defragmented my disk and created a larger swap file (Thank you
MathCAD 4 :(  )

System 386sx, 4MB, stacker 2.0, win31, DOS 5

---
--------------------------------------------------------------------------
1781
```

[67]:
```
vectorizer = sk_text.TfidfVectorizer(stop_words='english',
                                     #max_features = 1000,
                                     min_df=4, max_df=0.8)
data = vectorizer.fit_transform(news_data.data)
print(type(data))
```

```
<class 'scipy.sparse.csr.csr_matrix'>
```

[68]:
```
import sklearn.cluster as sk_cluster
k=3
kmeans = sk_cluster.KMeans(n_clusters=k, init='k-means++', max_iter=100,␣
 ↪n_init=1)
kmeans.fit_predict(data)
```

[68]:
```
array([2, 0, 0, …, 1, 2, 1])
```

To understand the clusters we can print the words that have the highest values in the centroid

[75]:
```
print("Top terms per cluster:")
asc_order_centroids = kmeans.cluster_centers_.argsort()#[:, ::-1]
order_centroids = asc_order_centroids[:,::-1]
terms = vectorizer.get_feature_names()
for i in range(k):
    print ("Cluster %d:" % i)
    for ind in order_centroids[i, :10]:
        print (' %s' % terms[ind])
    print
```

```
Top terms per cluster:
Cluster 0:
 year
 team
 game
 games
 runs
 baseball
 think
 good
 hit
 pitching
Cluster 1:
 space
 like
 just
 think
 nasa
 know
 don
 thanks
 does
 people
Cluster 2:
 windows
 file
 dos
 files
 drivers
 driver
 thanks
 card
 use
 problem
```

[69]:
```
C = metrics.confusion_matrix(kmeans.labels_,news_data.target)
```

```
mapped_kmeans_labels,C = cluster_class_mapping(kmeans.labels_,news_data.target)
print (C)
p = metrics.precision_score(news_data.target,mapped_kmeans_labels, average=None)
print(p)
r = metrics.recall_score(news_data.target,mapped_kmeans_labels, average = None)
print(r)
```

```
[[419   1   5]
 [170 596 384]
 [  2   0 204]]
[0.98588235 0.51826087 0.99029126]
[0.70896785 0.99832496 0.34401349]
```

[70]:
```
agglo = sk_cluster.AgglomerativeClustering(linkage = 'complete', n_clusters =␣
↪3,)
dense = data.todense()
agglo_labels = agglo.fit_predict(dense) # agglomerative needs dense data

C_agglo= metrics.confusion_matrix(agglo_labels,news_data.target)
print (C_agglo)
```

```
[[574 595 482]
 [ 17   0   2]
 [  0   2 109]]
```

[71]:
```
dbscan = sk_cluster.DBSCAN(eps=0.1)
dbscan_labels = dbscan.fit_predict(data)
C = metrics.confusion_matrix(dbscan.labels_,news_data.target)
print (C)
```

```
[[  0 556 567 576]
 [  0   9   0   0]
 [  0  26  30  17]
 [  0   0   0   0]]
```

[ ]:
```

```