Introduction-SciKit-Learn-Clustering

December 9, 2020

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.

[25]:
```
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:

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

Compute distance 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
```

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

Compute pairwise distances in a table

```
[31]: 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))
```

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

 all row distances
[0.5        0.5        1.         1.         0.         1.
 0.66666667 1.         0.66666667 1.        ]
[[0.         0.5        0.5        1.         1.        ]
 [0.5        0.         0.         1.         0.66666667]
 [0.5        0.         0.         1.         0.66666667]
 [1.         1.         1.         0.         1.        ]
 [1.         0.66666667 0.66666667 1.         0.        ]]
```

Compute distances using sklearn

[32]:
```
import sklearn.metrics as metrics

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

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

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\pairwise.py:1575:
DataConversionWarning: Data was converted to boolean for metric jaccard
  warnings.warn(msg, DataConversionWarning)
```

Compute distances between the rows of two tables

[34]:
```
B = np.random.randint(2, size = (3,3))
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.pairwise_distances(A,B,metric = 'jaccard')
print('\n the matrix of distances between the rows of A and B')
print(D3)
```

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

 the matrix of distances between the rows of A and B
[[0.         0.66666667 1.        ]
 [0.5        0.33333333 1.        ]
 [0.5        0.33333333 1.        ]
 [1.         1.         0.        ]
 [1.         0.33333333 1.        ]]
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\pairwise.py:1575:
DataConversionWarning: Data was converted to boolean for metric jaccard
  warnings.warn(msg, DataConversionWarning)
```

[35]:
```
sp_dist.cdist(A,B,'jaccard')
```

[35]:
```
array([[0.        , 0.66666667, 1.        ],
       [0.5       , 0.33333333, 1.        ],
       [0.5       , 0.33333333, 1.        ],
       [1.        , 1.        , 0.        ],
       [1.        , 0.33333333, 1.        ]])
```

We can apply everything to sparce matrices

[36]:
```
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:

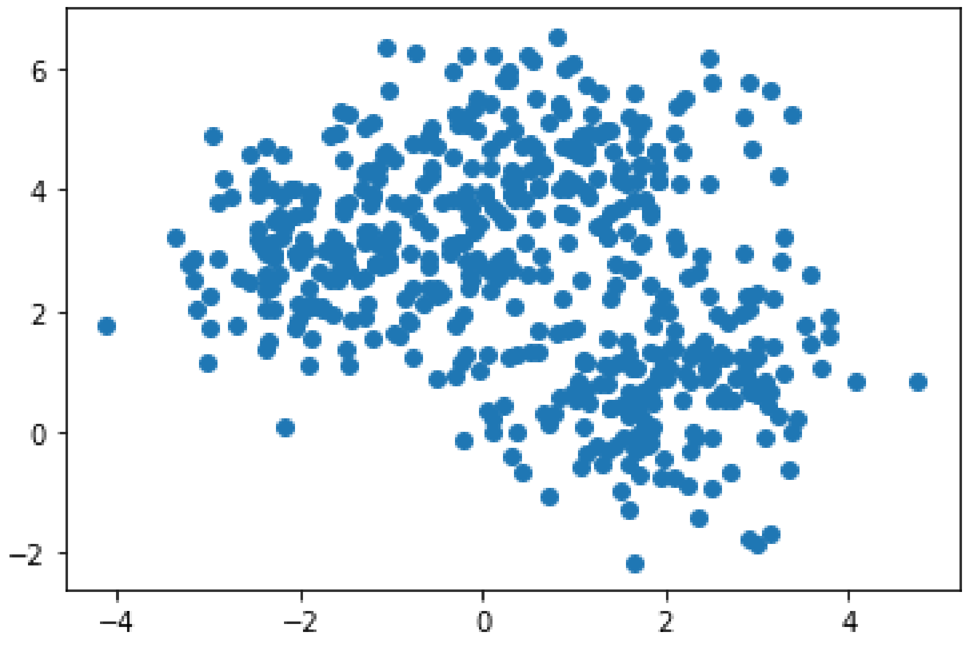
http://scikit-learn.org/stable/modules/clustering.html

Generate data from Gaussian distributions.

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

[37]:
```
centers = [[1,1], [-1, -1], [1, -1]]
X, true_labels = sk_data.make_blobs(n_samples=500, centers=3, n_features=2,
                                    center_box=(-10.0, 10.0),random_state=0)
plt.scatter(X[:,0], X[:,1])
```

```
[37]: <matplotlib.collections.PathCollection at 0x1a706d1c278>
```

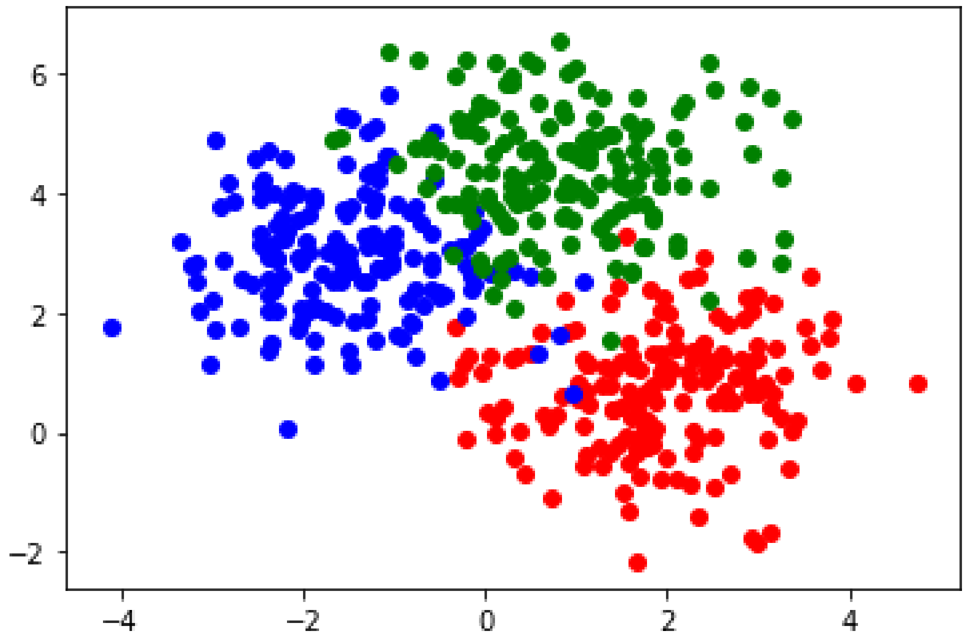


```
[38]: print(type(X))
      print(true_labels)
      print(len(true_labels[true_labels==0]),len(true_labels[true_labels==1]),len(true_labels[true_l
```

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

```
[39]: 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')
```

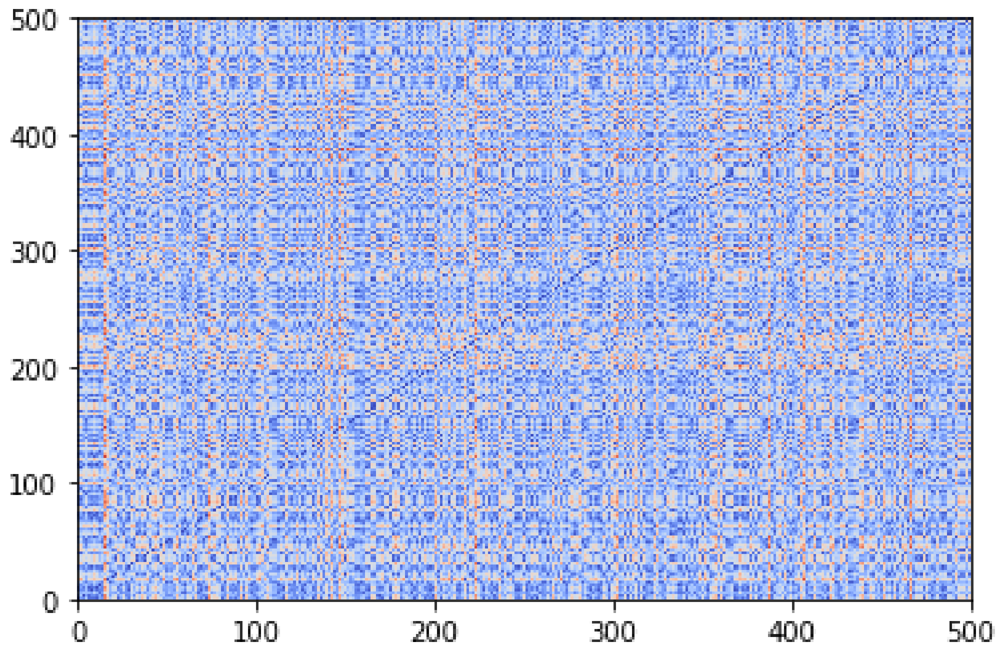
```
[39]: <matplotlib.collections.PathCollection at 0x1a706aad0b8>
```



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

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

```
[40]: <matplotlib.collections.QuadMesh at 0x1a706af4860>
```



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

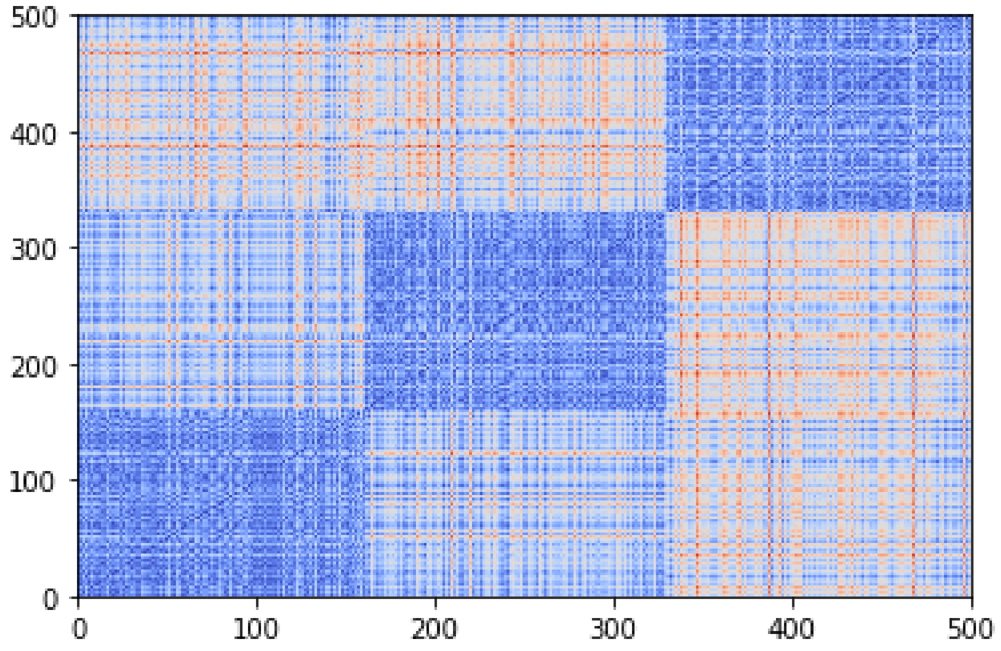
```
[41]: import sklearn.cluster as sk_cluster

      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:  881.748277486619

Cluster labels
[0 1 1 0 1 0 1 1 1 0 1 1 1 1 0 0 2 0 1 0 0 2 2 1 0 1 2 1 0 1 1 0 0 1 2 2 2
 2 2 2 1 0 0 2 2 0 2 0 1 1 0 0 0 0 1 1 2 2 0 2 2 1 2 2 2 1 2 2 1 0 1 0 0 1
 2 1 2 2 2 0 0 0 1 0 1 2 1 1 1 1 0 0 2 0 1 2 1 0 1 0 2 1 1 0 0 2 0 2 0 2 0
 0 2 1 1 2 1 2 2 1 1 2 0 1 1 0 1 0 1 0 2 0 1 2 0 2 1 1 2 0 0 0 1 2 0 0 0 2
 2 0 2 0 0 0 2 0 1 2 1 0 2 2 1 1 1 2 0 2 0 2 1 1 2 0 0 1 1 2 2 2 2 2 1 1 2
 2 0 2 1 1 1 0 2 2 0 1 2 0 1 2 1 2 0 0 1 2 2 2 1 1 0 2 2 1 1 0 1 0 1 1 2 1
 2 2 0 0 0 2 1 0 0 1 1 2 1 1 0 2 0 0 0 2 2 2 0 1 1 1 1 1 2 1 1 0 1 1 0 2 1
 2 2 0 0 2 2 0 1 2 1 2 2 1 2 0 1 2 0 2 2 2 2 1 2 0 0 0 0 1 1 1 2 1 1 0 0 2
 1 0 1 2 0 2 1 2 0 1 0 2 0 1 2 1 1 0 1 0 2 1 1 0 0 2 0 2 0 0 0 2 0 1 2 2 2
 1 2 2 0 1 1 1 2 1 2 0 0 0 2 0 1 1 2 0 1 1 1 2 2 2 2 1 0 2 1 0 0 0 2 0 1 0
 1 1 0 1 1 1 1 2 2 1 2 0 1 0 1 0 0 1 2 1 1 2 0 2 0 2 1 1 0 0 2 0 2 1 1 0 2
 1 2 1 0 0 1 2 2 0 0 2 1 1 0 1 2 0 1 0 2 0 1 0 0 2 0 0 0 0 2 0 2 1 2 1 0 1
 2 1 2 1 2 0 1 2 2 2 2 2 0 1 2 1 0 0 2 1 1 1 1 2 0 0 2 2 0 2 1 2 2 1 2 0 0
 2 2 0 2 1 1 0 2 2 1 0 1 0 0 2 2 2 0 2]

 Cluster Centroids
[[ 0.87564159  4.45514163]
 [-1.52371332  2.92068825]
 [ 1.96167358  0.73752985]]
```

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

```
[42]: 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)
```

```
[42]: <matplotlib.collections.QuadMesh at 0x1a706b47b70>
```



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

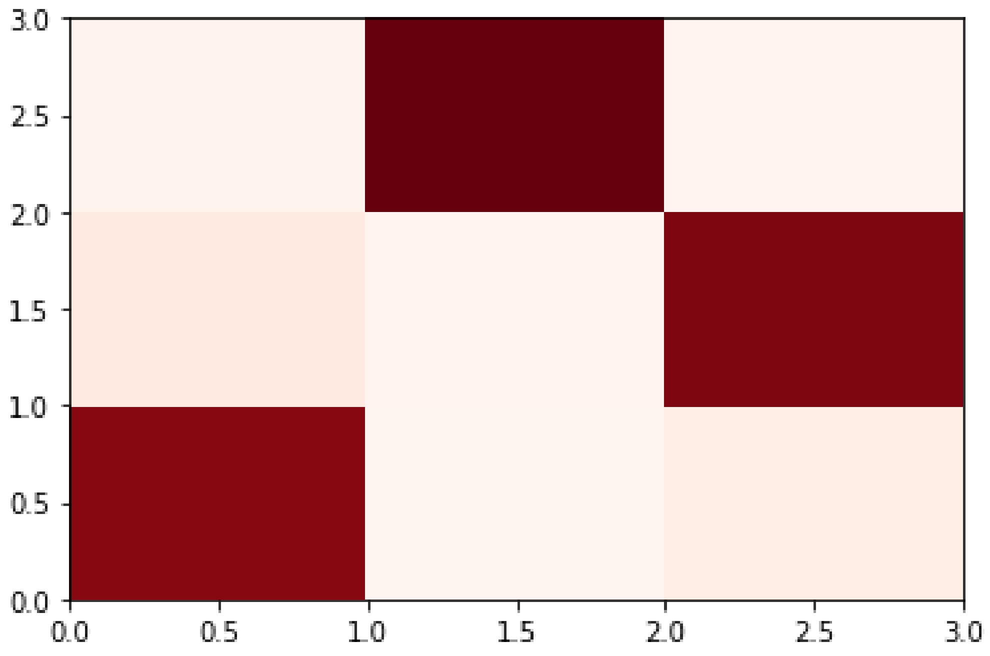
Recall: http://scikit-learn.org/stable/modules/generated/sklearn.metrics.recall_score.html#sklearn.metrics.recal

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

[43]:
```
C= metrics.confusion_matrix(kmeans_labels,true_labels)
print (C)
plt.pcolormesh(C,cmap=plt.cm.Reds)
```

```
[[151   2   9]
 [ 12   3 154]
 [  4 162   3]]
```

[43]:
```
<matplotlib.collections.QuadMesh at 0x1a705be89e8>
```



Compute precision and recall.

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

[44]:
```
p = metrics.precision_score(true_labels,kmeans_labels, average=None)
print(p)
r = metrics.recall_score(true_labels,kmeans_labels, average = None)
print(r)
```

```
[0.93209877 0.01775148 0.01775148]
[0.90419162 0.01796407 0.01807229]
```

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

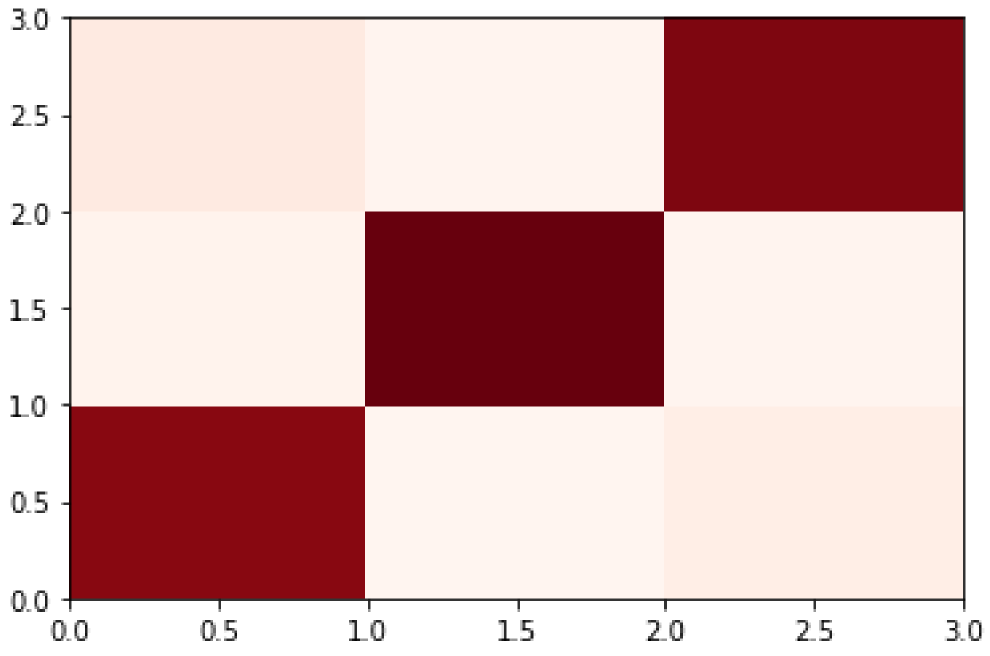
Useful command: numpy.argmax returns the index of the max element

[48]:
```
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)
```

```
[[151   2   9]
 [  4 162   3]
 [ 12   3 154]]
```

[48]:
```
<matplotlib.collections.QuadMesh at 0x1a705c9a9b0>
```



Compute different metrics for clustering quality

[49]:
```
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')
```

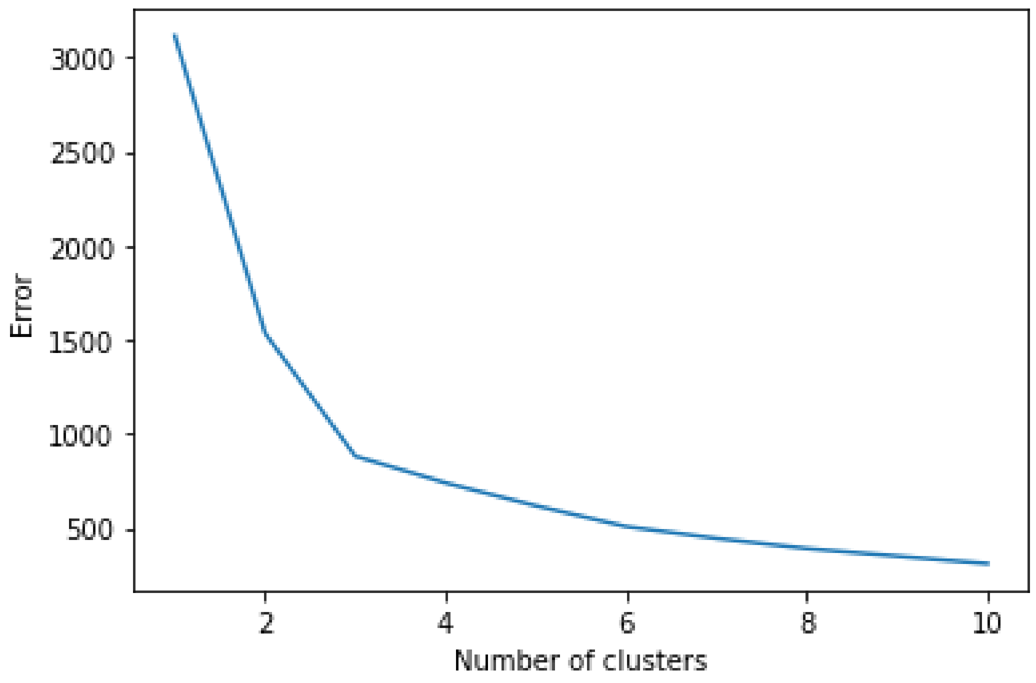
```
print(r)
f = metrics.f1_score(true_labels,mapped_kmeans_labels, average = 'weighted')
print(f)
```

```
0.7497037574992789
0.7498354394270633
0.7497695926813516
[0.93209877 0.95857988 0.9112426 ]
[0.90419162 0.97005988 0.92771084]
[0.91793313 0.96428571 0.91940299]
0.934019212506392
0.934
0.9339028852697001
```

The SSE plot

[50]:
```
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')
```

[50]:
```
Text(0, 0.5, 'Error')
```

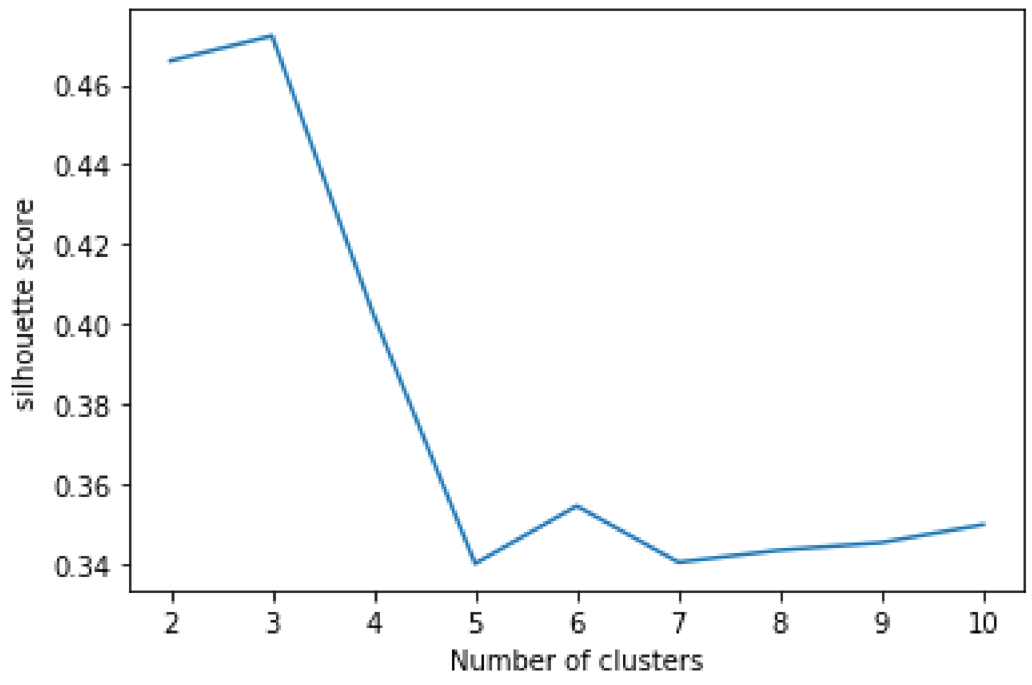


The silhouette plot

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

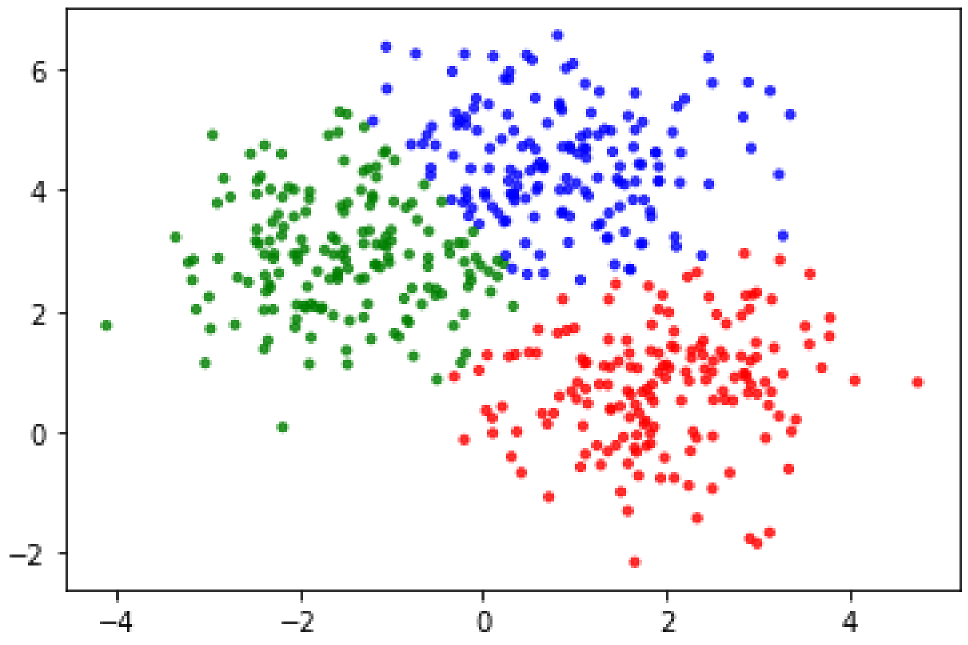
```
[51]: plt.plot(range(2,len(sh_score)),sh_score[2:])
      plt.xlabel('Number of clusters')
      plt.ylabel('silhouette score')
```

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



[52]:
```
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)
```

[52]:
```
<matplotlib.collections.PathCollection at 0x1a706b72400>
```

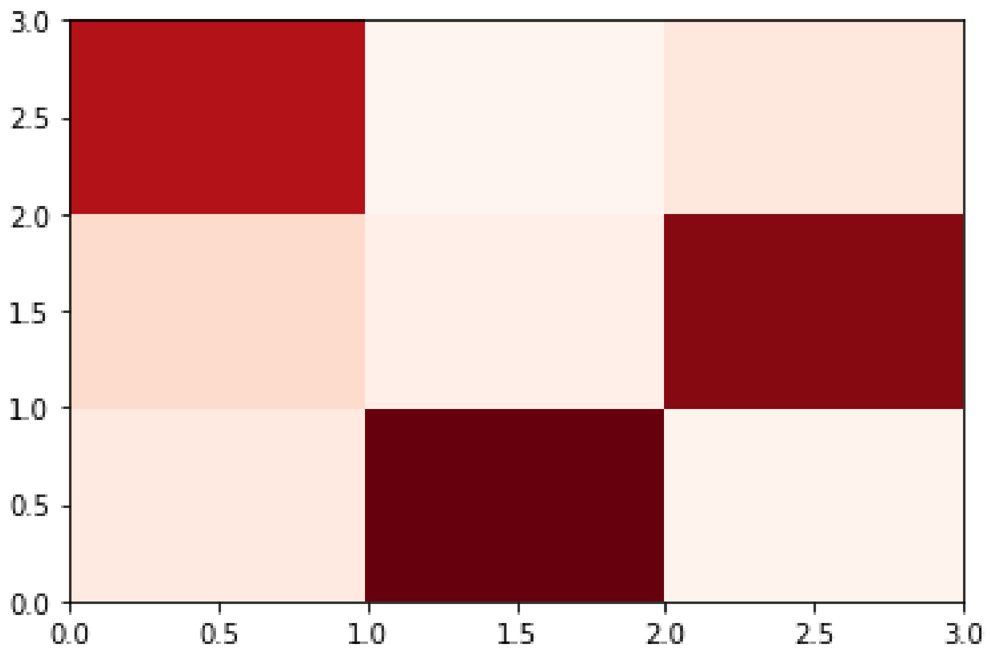


1.3.2 Agglomerative Clustering

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

[53]:
```
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)
```

```
[[ 12 159   3]
 [ 23   7 149]
 [132   1  14]]
[[132   1  14]
 [ 12 159   3]
 [ 23   7 149]]
0.8814828057981043
0.88
```



Another way to do agglomerative clustering using SciPy:

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

[54]:
```
import scipy.cluster.hierarchy as hr

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

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

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

[55]:
```
import scipy.spatial.distance as sp_dist
D = sp_dist.pdist(X, 'euclidean')
Z = hr.linkage(D, method='complete')
print (Z.shape, X.shape)
```

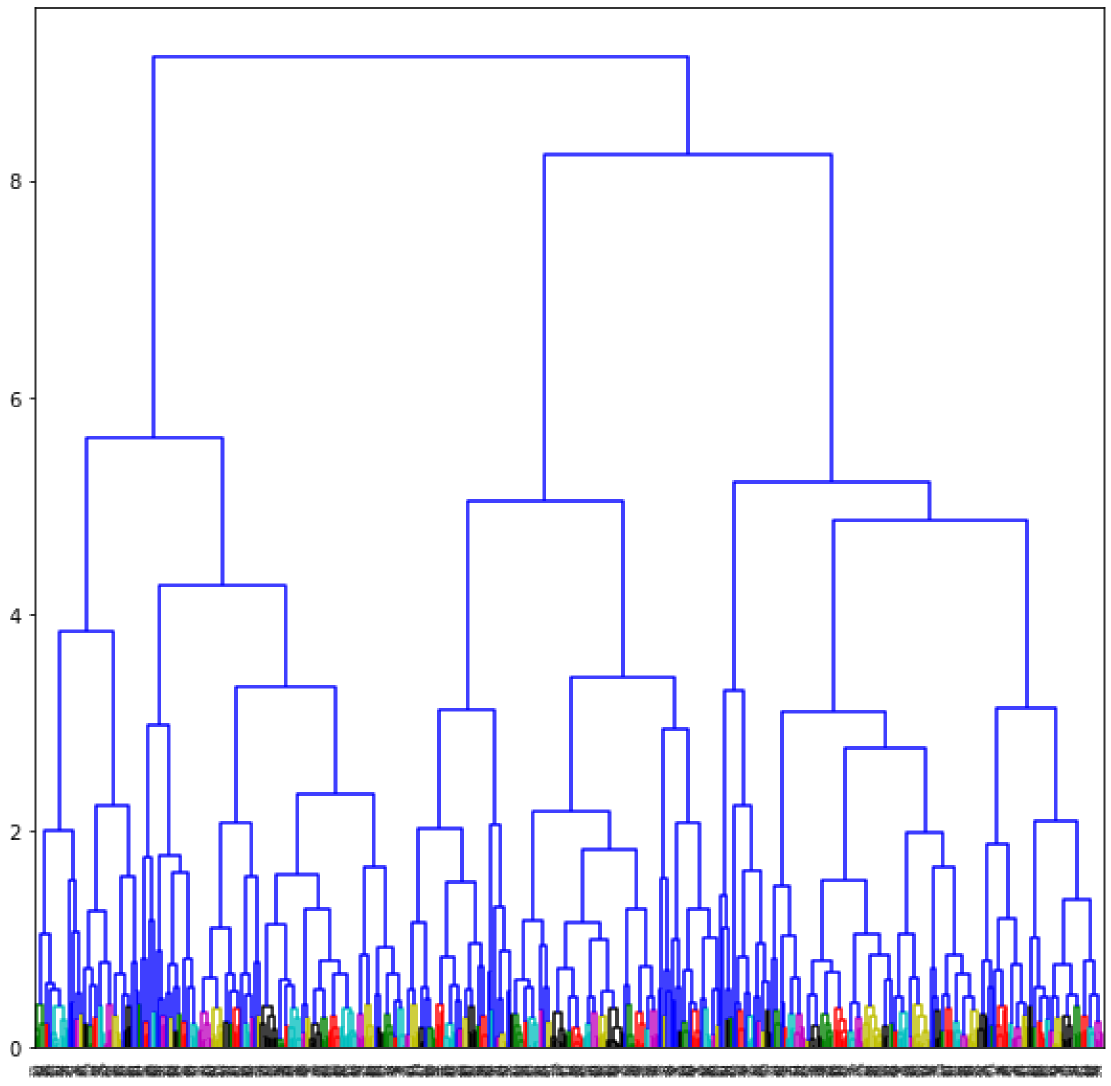
```
(499, 4) (500, 2)
```

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.

[56]:
```
fig = plt.figure(figsize=(10,10))
T = hr.dendrogram(Z,color_threshold=0.4, leaf_font_size=4)
```

```
fig.show()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:3: UserWarning:
Matplotlib is currently using module://ipykernel.pylab.backend_inline, which is
a non-GUI backend, so cannot show the figure.
  This is separate from the ipykernel package so we can avoid doing imports
until
```



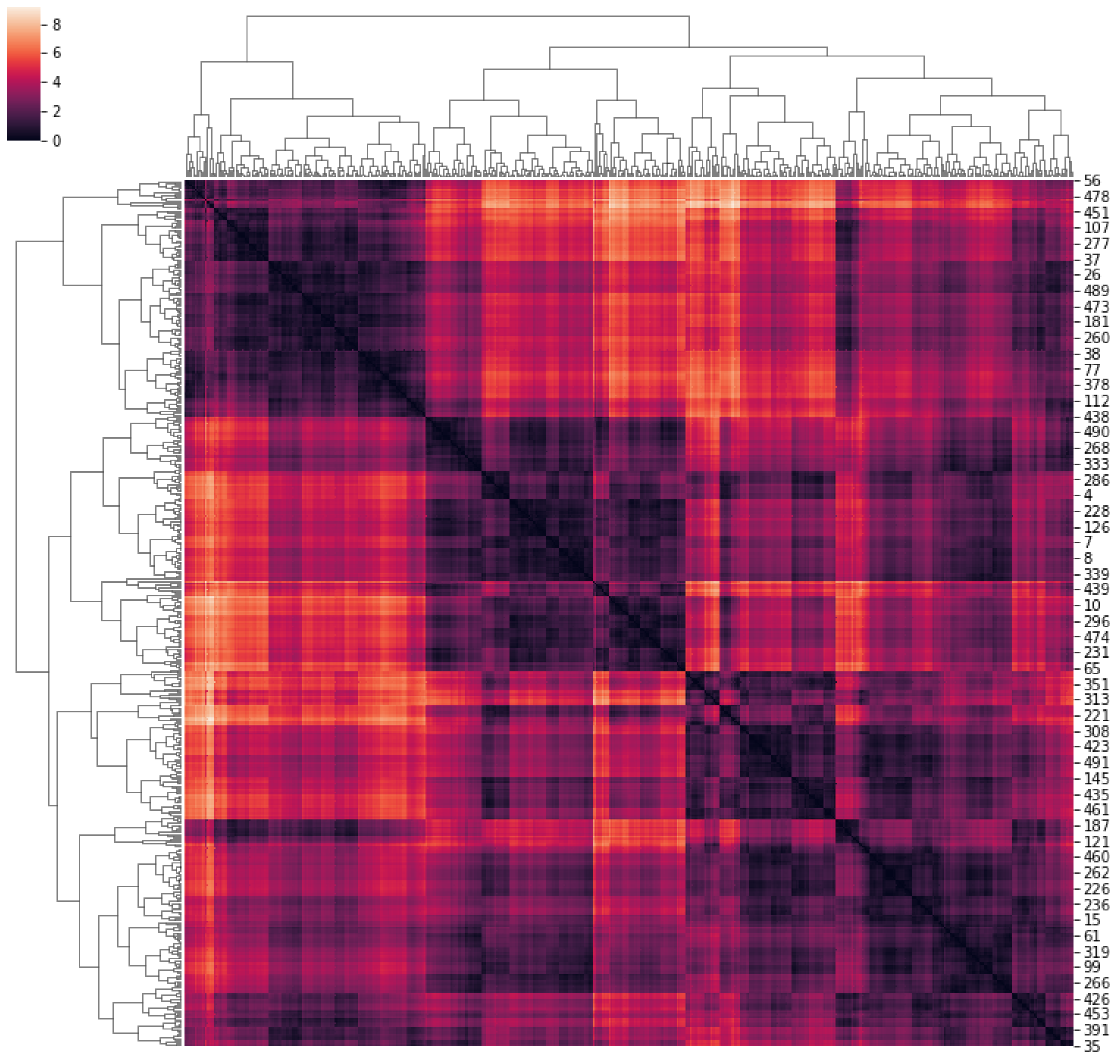
Another way to do agglomerative clustering (and visualizing it): http://seaborn.pydata.org/generated/seaborn.clustermap.html

[57]:
```
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:603:
ClusterWarning: scipy.cluster: The symmetric non-negative hollow observation
matrix looks suspiciously like an uncondensed distance matrix
  metric=self.metric)
```

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



1.3.3 DBSCAN Algorithm

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

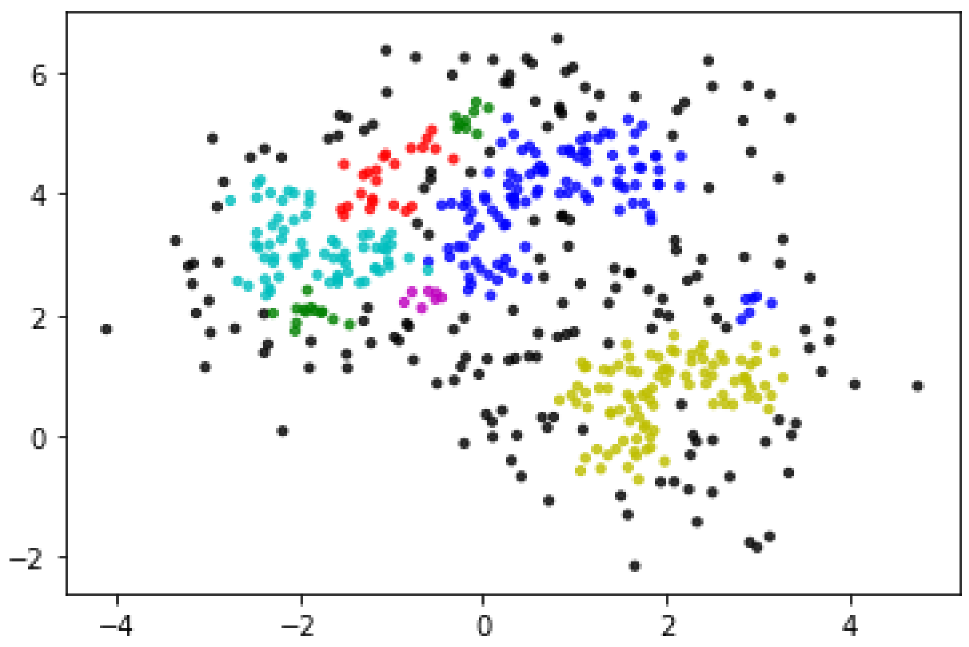
[58]:
```
dbscan = sk_cluster.DBSCAN(eps=0.3)
dbscan_labels = dbscan.fit_predict(X)
print(dbscan_labels) #label -1 corresponds to noise
renamed_dbscan_labels = [x+1 for x in dbscan_labels]
C = metrics.confusion_matrix(renamed_dbscan_labels,true_labels)
#print(C)
print (C[:,:max(true_labels)+1])
```

```
[ 0  3  1 -1  2  0 -1  2  3  6  3  2 -1  4  0 -1 -1 -1  3  0 -1  5  5  3
  6  7  5 -1  0  3  7 -1  9 -1  5 -1  5  5  5 -1  3  0 -1  8 -1 -1  5 -1
  3 -1  0  0  0  0 -1  1 -1  5  0 -1 -1  0  5  5 -1 -1 -1  5  0  0  3 -1
```

```
 -1  3 -1  2  5  5  5 -1 -1 -1  3  0 -1 -1  3 -1  4  3  0  0 -1 -1  0  5
 -1  0  0  0 -1  3 -1  0 -1 -1  0 -1  0  5  0  0 -1  0  1  5  2  5  5  7
  1 -1  0 -1  3  6  3  0 -1 -1  5  0  3  5  0  5  3  3 -1 -1 -1  2  3 -1
 -1  0  0 -1 -1 -1  5 -1 -1  0  8 -1  0 -1  0 -1  5 -1  1  1  3  5  0  5
  2  5  2  3  5  0  0  2  3  5  5 -1  5  5  3 -1  5 -1  0  8  0  3  3  0
  5  5  0  3  5  0  3  5  2 -1  6  9  3 -1 -1 -1 -1  3  0 -1 -1  3  3  0
 -1 -1  1  4  5 -1 -1 -1 -1 -1  0  5  3 -1  0  3  0 -1  2  3  6  5  0  0
 -1  5  5  5  0  3  3  3  3  1  5  3  3  0 -1  3  9 -1  3 -1  5  0  0  5
  5  0  0 -1  7  5 -1 -1  5  0  0  5  0  5 -1  5 -1 -1 -1  0 -1 -1  2  7
  3 -1 -1  1  3 -1  0  5  3  0  3 -1 -1  5  4 -1 -1  3  9  5  0  2  5  4
 -1 -1  0  0  5  3  3  0 -1  5  0  5 -1 -1  6  5  0  1  8 -1  5  4  5 -1
  9  4  0  2  5  3  5 -1  0  0 -1  0 -1 -1  5 -1 -1  3  3 -1  5  5 -1 -1
  0 -1  3  2  0  0 -1  0  3  0  3 -1 -1  3  1  0  2  5  5  1  5 -1  3 -1
  3  0 -1  0 -1  2  0 -1  0  5 -1 -1  2 -1  6 -1  5 -1  5 -1  3 -1  5 -1
  5  0  9  2 -1  5  5  0  9  5  3 -1  0  3 -1  0 -1  0 -1  0 -1 -1 -1  5
  6  0  0  0  5  0 -1 -1 -1 -1 -1 -1  5  2  5 -1 -1  0 -1 -1 -1 -1 -1  5
  0  2  5  3  0 -1 -1  2 -1  0  3 -1 -1  9  5 -1  0  5  3 -1  5 -1 -1 -1
  0  5 -1  2  5  3  2  0  5  5  1  0 -1  0 -1 -1  8  8  9  5]
[[58 68 41]
 [87  0 15]
 [ 0  0 13]
 [ 6  0 18]
 [ 0  0 66]
 [ 0  0  7]
 [ 0 92  1]
 [ 7  1  0]
 [ 0  0  5]
 [ 0  6  0]
 [ 9  0  0]]
```

[59]:
```
#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)
```

[59]:
```
<matplotlib.collections.PathCollection at 0x1a7176931d0>
```



2 Processing Complex Data

So far we have assumed that the intput is in the form of numerical vectors to which we can apply directly the algorithms we have. Often the data will be more complex. For example what if we want to cluster categorical data, itemsets, or text? Python provides libraries for processing the data and transforming them to a format that we can use.

Python offers a set of tools for extracting features:http://scikit-learn.org/stable/modules/feature_extraction.html

2.0.1 DictVectorizer

The DictVectorizer feature extraction: http://scikit-learn.org/stable/modules/generated/sklearn.feature_extractio

The DictVectorizer takes a dictionary of attribute-value pairs and transforms them into numerical vectors. Real values are preserved, while categorical attributes are transformed into binary. The vectorizer produces a *sparse representation*.

[61]:
```
from sklearn.feature_extraction import DictVectorizer

measurements = [
{'city': 'Dubai', 'temperature': 45},
{'city': 'London', 'temperature': 12},
{'city': 'San Fransisco', 'temperature': 23},
]
vec = DictVectorizer()
print(type(vec.fit_transform(measurements)))
print(vec.fit_transform(measurements).toarray())
vec.get_feature_names()
```

```
<class 'scipy.sparse.csr.csr_matrix'>
[[ 1.  0.  0. 45.]
 [ 0.  1.  0. 12.]
 [ 0.  0.  1. 23.]]
```

[61]:
```
['city=Dubai', 'city=London', 'city=San Fransisco', 'temperature']
```

[66]:
```
from sklearn.feature_extraction import DictVectorizer

measurements = [
{'city': 'Dubai', 'temperature': 45, 'dummy': 3},
{'city': 'London', 'temperature': 12},
{'city': 'San Fransisco', 'temperature': 23},
]
vec = DictVectorizer()
vec.fit(measurements)
print(vec.get_feature_names())
print(vec.transform(measurements).toarray())
x = {'city': 'Athens', 'temperature': 32, 'dummy2': 2}
print(vec.transform(x).toarray())
```

```
['city=Dubai', 'city=London', 'city=San Fransisco', 'dummy', 'temperature']
[[ 1.  0.  0.  3. 45.]
 [ 0.  1.  0.  0. 12.]
 [ 0.  0.  1.  0. 23.]]
[[ 0.  0.  0.  0. 32.]]
```

[67]:
```
measurements = [
    {'refund' : 'No','marital_status': 'married', 'income' : 100},
    {'refund' : 'Yes','marital_status': 'single', 'income' : 120},
    {'refund' : 'No','marital_status':'divorced', 'income' : 80},
]
vec = DictVectorizer()
print(vec.fit_transform(measurements))
vec.get_feature_names()
```

```
  (0, 0)        100.0
  (0, 2)        1.0
  (0, 4)        1.0
  (1, 0)        120.0
  (1, 3)        1.0
  (1, 5)        1.0
  (2, 0)        80.0
  (2, 1)        1.0
  (2, 4)        1.0
```

[67]:
```
['income',
 'marital_status=divorced',
 'marital_status=married',
 'marital_status=single',
 'refund=No',
 'refund=Yes']
```

2.1 Text processing

Feature extraction from text: http://scikit-learn.org/stable/modules/classes.html#text-feature-extraction-ref

2.1.1 CountVectorizer

The CountVectorizer can be used to extract features in the form of bag of words. It is typically used for text, but you could use it to represent also a collection of itemsets (where each itemset will become a word).

[68]:
```python
import sklearn.feature_extraction.text as sk_text

corpus = ['This is the first document.',
           'this is the second second document.',
           'And the third one.',
           'Is this the first document?',
          ]

vectorizer = sk_text.CountVectorizer(min_df=1)
X = vectorizer.fit_transform(corpus)
print(X.toarray())
vectorizer.get_feature_names()
```

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

[68]:
```
['and', 'document', 'first', 'is', 'one', 'second', 'the', 'third', 'this']
```

[19]:
```python
vectorizer = sk_text.CountVectorizer(min_df=1,stop_words = 'english')

X2 = vectorizer.fit_transform(corpus)
print(X2.toarray())
vectorizer.get_feature_names()
```

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

```
[19]: ['document', 'second']
```

2.1.2 TfIdfVectorizer

TfIdfVectorizer transforms text into a sparse matrix where rows are text and columns are words, and values are the tf-dif values. It performs tokenization, normalization, and removes stop-words. More here: http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html#sklearn.feature_extrac

```
[69]: vectorizer = sk_text.TfidfVectorizer(min_df=1)
X = vectorizer.fit_transform(corpus)
print(X.toarray())
print (vectorizer.get_feature_names())
```

```
[[0.         0.43877674 0.54197657 0.43877674 0.         0.
  0.35872874 0.         0.43877674]
 [0.         0.27230147 0.         0.27230147 0.         0.85322574
  0.22262429 0.         0.27230147]
 [0.55280532 0.         0.         0.         0.55280532 0.
  0.28847675 0.55280532 0.        ]
 [0.         0.43877674 0.54197657 0.43877674 0.         0.
  0.35872874 0.         0.43877674]]
['and', 'document', 'first', 'is', 'one', 'second', 'the', 'third', 'this']
```

Removing stop-words

```
[37]: vectorizer = sk_text.TfidfVectorizer(stop_words = 'english',min_df=1)
X = vectorizer.fit_transform(corpus)
print(X.toarray())
print (vectorizer.get_feature_names())
```

```
[[1.         0.        ]
 [0.30403549 0.9526607 ]
 [0.         0.        ]
 [1.         0.        ]]
['document', 'second']
```

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

```
[71]: 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
```

[72]:
```
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\\tsapa\\scikit_learn_data\\20news_home\\20news-bydate-
train\\sci.space\\60940'
 'C:\\Users\\tsapa\\scikit_learn_data\\20news_home\\20news-bydate-
train\\comp.os.ms-windows.misc\\9955'
 'C:\\Users\\tsapa\\scikit_learn_data\\20news_home\\20news-bydate-
train\\comp.os.ms-windows.misc\\9846'
 …
 'C:\\Users\\tsapa\\scikit_learn_data\\20news_home\\20news-bydate-
train\\sci.space\\60891'
 'C:\\Users\\tsapa\\scikit_learn_data\\20news_home\\20news-bydate-
train\\rec.sport.baseball\\104484'
 'C:\\Users\\tsapa\\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 autoexec.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
```

[73]:
```
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'>
```

2.2 Clustering text data

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

[74]:
```
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)
```

[74]:
```
array([1, 2, 2, …, 1, 1, 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
```

[76]:
```
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)
```

```
[[361   0   0]
 [  0 364   2]
 [230 233 591]]
[1.         0.99453552 0.56072106]
[0.6108291  0.60971524 0.99662732]
```

[79]:
```
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]]
```

[81]:
```
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]]
```

2.3 Feature normalization

Python provides some functionality for normalizing and standardizing the data. Be careful though, some operations work only with dense data.

http://scikit-learn.org/stable/modules/preprocessing.html#preprocessing

Use the function **preprocessing.scale** to normalize by removing the mean and dividing by the standard deviation. This is done per **feature**, that is, per column of the dataset.

[82]:
```
from sklearn import preprocessing

X = np.array([[ 1., -1.,  2.],
              [ 2.,  0.,  1.],
              [ 0.,  1., -1.]])
print("column means: ",X.mean(axis = 0))
print("column std: ",X.std(axis = 0))
X_scaled = preprocessing.scale(X)
print("after feature normalization")
print(X_scaled)
print("normalized column means: ",X_scaled.mean(axis=0))
print("normalized column std: ",X_scaled.var(axis = 0))
```

```
column means:  [1.         0.         0.66666667]
column std:  [0.81649658 0.81649658 1.24721913]
after feature normalization
[[ 0.         -1.22474487  1.06904497]
 [ 1.22474487  0.          0.26726124]
 [-1.22474487  1.22474487 -1.33630621]]
normalized column means:  [0.00000000e+00 0.00000000e+00 1.48029737e-16]
normalized column std:  [1. 1. 1.]
```

Feature normalization will not work with sparse data. In this case, the zeros are treated as values, so the sparse matrix will become non-sparse after normalization.

[83]:
```
import scipy.sparse
cX = scipy.sparse.csc_matrix(X)
cX_scaled = preprocessing.scale(cX)
print(cX_scaled)
```

```
      ␣
↪---------------------------------------------------------------------------

       ValueError                                Traceback (most recent call␣
↪last)
```

```
<ipython-input-83-961e7864f1cd> in <module>
      1 import scipy.sparse
      2 cX = scipy.sparse.csc_matrix(X)
----> 3 cX_scaled = preprocessing.scale(cX)
      4 print(cX_scaled)


C:\ProgramData\Anaconda3\lib\site-packages\sklearn\preprocessing\data.py␣
↪in scale(X, axis, with_mean, with_std, copy)
    143         if with_mean:
    144             raise ValueError(
--> 145                 "Cannot center sparse matrices: pass␣
↪`with_mean=False` instead"
    146                 " See docstring for motivation and alternatives.")
    147         if axis != 0:


ValueError: Cannot center sparse matrices: pass `with_mean=False`␣
↪instead See docstring for motivation and alternatives.
```

The same can be done with the **StandardScaler** from the preprocessing library of sklearn.

The function **fit()** computes the parameters for scaling, and **transform()** applies the scaling

[84]:
```python
from sklearn import preprocessing
std_scaler = preprocessing.StandardScaler()
std_scaler.fit(X)
print(std_scaler.mean_)
print(std_scaler.scale_)
X_std = std_scaler.transform(X)
print("scaled data:")
print(X_std)
```

```
[1.         0.         0.66666667]
[0.81649658 0.81649658 1.24721913]
scaled data:
[[ 0.         -1.22474487  1.06904497]
 [ 1.22474487  0.          0.26726124]
 [-1.22474487  1.22474487 -1.33630621]]
```

The advantage is the we can now apply the transform to new data.

For example, we compute the parameters for the training data and we apply the scaling to the test data.

[85]:
```python
y = np.array([[2.,3.,1.],
              [1.,2.,1.]])
print(std_scaler.transform(y))
```

```
[[1.22474487 3.67423461 0.26726124]
 [0.         2.44948974 0.26726124]]
```

The **MinMaxScaler** subbtracts from each column the minimum and then divides by the max-min.

[86]:
```
min_max_scaler = preprocessing.MinMaxScaler()
X_minmax = min_max_scaler.fit_transform(X)
print(X_minmax)
print(min_max_scaler.transform(y))
```

```
[[0.5        0.         1.        ]
 [1.         0.5        0.66666667]
 [0.         1.         0.        ]]
[[1.         2.         0.66666667]
 [0.5        1.5        0.66666667]]
```

The **MaxAbsScaler** divides with the maximum absolute value.

The MaxAbsScaler can work with sparse data, since it does not destroy the data sparseness. For the other datasets, removing the mean (or min) can destroy the sparseness of the data.

Sometimes we may choose to normalize only the non-zero values. This should be done manually.

[87]:
```
max_abs_scaler = preprocessing.MaxAbsScaler()
X_maxabs = max_abs_scaler.fit_transform(X)
X_maxabs
```

[87]:
```
array([[ 0.5, -1. ,  1. ],
       [ 1. ,  0. ,  0.5],
       [ 0. ,  1. , -0.5]])
```

[88]:
```
# works with sparse data
cX_scaled = max_abs_scaler.transform(cX)
print(cX_scaled)
```

```
  (0, 0)        0.5
  (1, 0)        1.0
  (0, 1)        -1.0
  (2, 1)        1.0
  (0, 2)        1.0
  (1, 2)        0.5
  (2, 2)        -0.5
```

The **normalize** function normalizes the **rows** so that they become unit vectors in some norm that we specify. It can be applied to sparse matrices without destroying the sparsity.

[89]:
```
#works with sparse data

X_normalized = preprocessing.normalize(X, norm='l2')

X_normalized
```

```
[89]: array([[ 0.40824829, -0.40824829,  0.81649658],
              [ 0.89442719,  0.        ,  0.4472136 ],
              [ 0.        ,  0.70710678, -0.70710678]])
```

```
[90]: crX = scipy.sparse.csr_matrix(X)
      crX_scaled = preprocessing.normalize(crX,norm='l1')
      print(crX_scaled)
```

```
  (0, 0)	0.25
  (0, 1)	-0.25
  (0, 2)	0.5
  (1, 0)	0.6666666666666666
  (1, 2)	0.3333333333333333
  (2, 1)	0.5
  (2, 2)	-0.5
```

2.3.1 OneHotEncoder

The **OneHotEncoder** can be used for categorical data to transform them into binary, where for each attribute value we have 0 or 1 depending on whether this value appears in the feature vector. It works with numerical categorical values.

```
[91]: X = [[0,1,2],
           [1,2,3],
           [0,1,4]]
      enc = preprocessing.OneHotEncoder(handle_unknown='ignore')
      enc.fit(X)
      enc.transform([[0,2,4],[1,1,2]]).toarray()
```

```
[91]: array([[1., 0., 0., 1., 0., 0., 1.],
             [0., 1., 1., 0., 1., 0., 0.]])
```

In this example every number in every column defines a separate feature

```
[92]: enc.categories_
```

```
[92]: [array([0, 1]), array([1, 2]), array([2, 3, 4])]
```

We can also apply it selectively to some columns of the data

```
[93]: #works with sparse data

      X = np.array([[0, 10, 45100],
           [1, 20, 45221],
           [0, 20, 45212]])
      enc = preprocessing.OneHotEncoder(categorical_features=[2]) #only the third␣
       ↪column is categorical
      enc.fit(X)
enc.transform([[5,13,45212],[4,12,45221]]).toarray()
```

```
C:\ProgramData\Anaconda3\lib\site-
packages\sklearn\preprocessing\_encoders.py:415: FutureWarning: The handling of
integer data will change in version 0.22. Currently, the categories are
determined based on the range [0, max(values)], while in the future they will be
determined based on the unique values.
If you want the future behaviour and silence this warning, you can specify
"categories='auto'".
In case you used a LabelEncoder before this OneHotEncoder to convert the
categories to integers, then you can now use the OneHotEncoder directly.
  warnings.warn(msg, FutureWarning)
C:\ProgramData\Anaconda3\lib\site-
packages\sklearn\preprocessing\_encoders.py:451: DeprecationWarning: The
'categorical_features' keyword is deprecated in version 0.20 and will be removed
in 0.22. You can use the ColumnTransformer instead.
  "use the ColumnTransformer instead.", DeprecationWarning)
```

```
[93]: array([[ 0.,  1.,  0.,  5., 13.],
              [ 0.,  0.,  1.,  4., 12.]])
```

```
[ ]:
```