Jörn's Blog

Science, code and stuff...

SciPy Hierarchical Clustering and Dendrogram Tutorial

This is a tutorial on how to use scipy's hierarchical clustering.

One of the benefits of hierarchical clustering is that you don't need to already know the number of clusters k in your data in advance. Sadly, there doesn't seem to be much documentation on how to actually use scipy's hierarchical clustering to make an informed decision and then retrieve the clusters.

In the following I'll explain:

- how to use scipy's hierarchical clustering
- how to plot a nice dendrogram from it
- how to use the dendrogram to select a distance cut-off (aka determining the number of clusters k in your data)
- how to retrieve the k clusters
- how to visualize the clusters (2D case)

Other works:

Some short shameless self-advertising:

- I teach machines to associate like humans. In that project I used hierarchical clustering to group similar learned graph patterns together.
- I'm always searching for good students, contact me.

Naming conventions:

Before we start, as i know that it's easy to get lost, some naming conventions:

- X samples (n x m array), aka data points or "singleton clusters"
- n number of samples
- m number of features
- Z cluster linkage array (contains the hierarchical clustering information)
- k number of clusters

So, let's go.

Imports and Setup

In [1]:

```
# needed imports
from matplotlib import pyplot as plt
from scipy.cluster.hierarchy import dendrogram, linkage
import numpy as np
```

In [2]:

```
# some setting for this notebook to actually show the graphs inline
# you probably won't need this
%matplotlib inline
np.set_printoptions(precision=5, suppress=True)  # suppress scientific float notation
```

Generating Sample Data

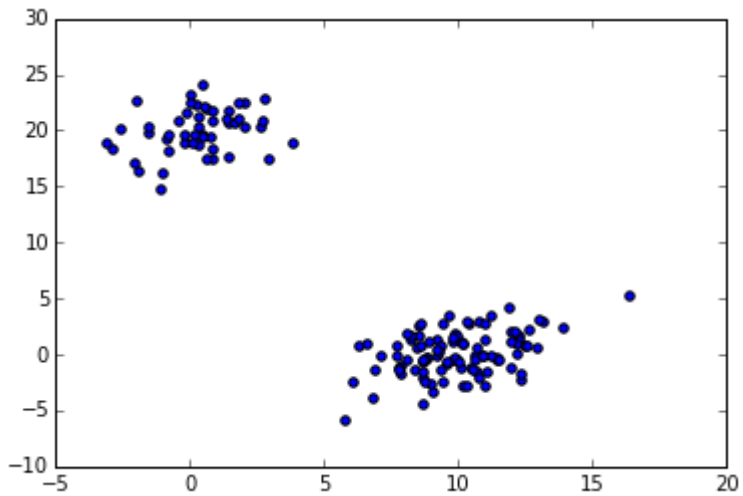
You'll obviously not need this step to run the clustering if you have own data.

The only thing you need to make sure is that you convert your data into a matrix `X` with `n` samples and `m` features, so that `X.shape == (n, m)`.

In [3]:

```
# generate two clusters: a with 100 points, b with 50:
np.random.seed(4711)  # for repeatability of this tutorial
a = np.random.multivariate_normal([10, 0], [[3, 1], [1, 4]], size=[100,])
b = np.random.multivariate_normal([0, 20], [[3, 1], [1, 4]], size=[50,])
X = np.concatenate((a, b),)
print X.shape  # 150 samples with 2 dimensions
plt.scatter(X[:,0], X[:,1])
plt.show()
```

```
(150, 2)
```



Perform the Hierarchical Clustering

Now that we have some very simple sample data, let's do the actual clustering on it:

In [4]:

```
# generate the linkage matrix
Z = linkage(X, 'ward')
```

Done. That was pretty simple, wasn't it?

Well, sure it was, this is python ;), but what does the weird `'ward'` mean there and how does this actually work?

As the scipy `linkage` docs tell us, `'ward'` is one of the methods that can be used to calculate the distance between newly formed clusters. `'ward'` causes `linkage()` to use the Ward variance minimization algorithm.

I think it's a good default choice, but it never hurts to play around with some other common linkage methods like `'single'`, `'complete'`, `'average'`, ... and the different distance metrics like `'euclidean'` (default), `'cityblock'` aka Manhattan, `'hamming'`, `'cosine'`... if you have the feeling that your data should not just be clustered to minimize the overall intra cluster variance in euclidean space. For example, you should have such a weird feeling with long (binary) feature vectors (e.g., word-vectors in text clustering).

As you can see there's a lot of choice here and while python and scipy make it very easy to do the clustering, it's you who has to understand and make these choices. If i find the time, i might give some more practical advice about this, but for now i'd urge you to at least read up on the linked methods and metrics to make a somewhat informed choice. Another thing you can and should definitely do is check the Cophenetic Correlation Coefficient of your clustering with help of the `cophenet()` function. This (very very briefly) compares (correlates) the actual pairwise distances of all your samples to those implied by the hierarchical clustering. The closer the value is to 1, the better the clustering preserves the original distances, which in our case is pretty close:

In [5]:

```
from scipy.cluster.hierarchy import cophenet
from scipy.spatial.distance import pdist

c, coph_dists = cophenet(Z, pdist(X))
c
```

Out[5]:

```
0.98001483875742679
```

No matter what `method` and `metric` you pick, the `linkage()` function will use that method and metric to calculate the distances of the clusters (starting with your `n` individual samples (aka data points) as singleton clusters)) and in each iteration will merge the two clusters which have the smallest distance according the selected `method` and `metric`. It will return an array of length `n - 1` giving you information about the `n - 1` cluster merges which it needs to pairwise merge `n` clusters. `Z[i]` will tell us which clusters were merged in the `i`-th iteration, let's take a look at the first two points that were merged:

In [6]:

```
Z[0]
```

Out[6]:

```
array([ 52.     ,  53.     ,   0.04151,   2.     ])
```

We can see that ach row of the resulting array has the format `[idx1, idx2, dist, sample_count]`.

In its first iteration the `linkage` algorithm decided to merge the two clusters (original samples here) with indices 52 and 53, as they only had a distance of 0.04151. This created a cluster with a total of 2 samples.

In [7]:

```
Z[1]
```

Out[7]:

```
array([ 14.     ,  79.     ,   0.05914,   2.     ])
```

In the second iteration the algorithm decided to merge the clusters (original samples here as well) with indices 14 and 79, which had a distance of 0.04914. This again formed another cluster with a total of 2 samples.

The indices of the clusters until now correspond to our samples. Remember that we had a total of 150 samples, so indices 0 to 149. Let's have a look at the first 20 iterations:

In [8]:

```
Z[:20]
```

Out[8]:

```
array([[  52.    ,   53.    ,    0.04151,    2.    ],
       [  14.    ,   79.    ,    0.05914,    2.    ],
       [  33.    ,   68.    ,    0.07107,    2.    ],
       [  17.    ,   73.    ,    0.07137,    2.    ],
       [   1.    ,    8.    ,    0.07543,    2.    ],
       [  85.    ,   95.    ,    0.10928,    2.    ],
       [ 108.    ,  131.    ,    0.11007,    2.    ],
       [   9.    ,   66.    ,    0.11302,    2.    ],
       [  15.    ,   69.    ,    0.11429,    2.    ],
       [  63.    ,   98.    ,    0.1212 ,    2.    ],
       [ 107.    ,  115.    ,    0.12167,    2.    ],
       [  65.    ,   74.    ,    0.1249 ,    2.    ],
       [  58.    ,   61.    ,    0.14028,    2.    ],
       [  62.    ,  152.    ,    0.1726 ,    3.    ],
       [  41.    ,  158.    ,    0.1779 ,    3.    ],
       [  10.    ,   83.    ,    0.18635,    2.    ],
       [ 114.    ,  139.    ,    0.20419,    2.    ],
       [  39.    ,   88.    ,    0.20628,    2.    ],
       [  70.    ,   96.    ,    0.21931,    2.    ],
       [  46.    ,   50.    ,    0.22049,    2.    ]])
```

We can observe that until iteration 13 the algorithm only directly merged original samples. We can also observe the monotonic increase of the distance.

In iteration 13 the algorithm decided to merge cluster indices 62 with 152. If you paid attention the 152 should astonish you as we only have original sample indices 0 to 149 for our 150 samples. All indices `idx >= len(X)` actually refer to the cluster formed in `Z[idx - len(X)]`.

This means that while `idx` 149 corresponds to `X[149]` that `idx` 150 corresponds to the cluster formed in `Z[0]`, `idx` 151 to `Z[1]`, 152 to `Z[2]`, ...

Hence, the merge iteration 13 merged sample 62 to our samples 33 and 68 that were previously merged in iteration 2 (152 - 2).

Let's check out the points coordinates to see if this makes sense:

In [9]:

```
X[[33, 68, 62]]
```

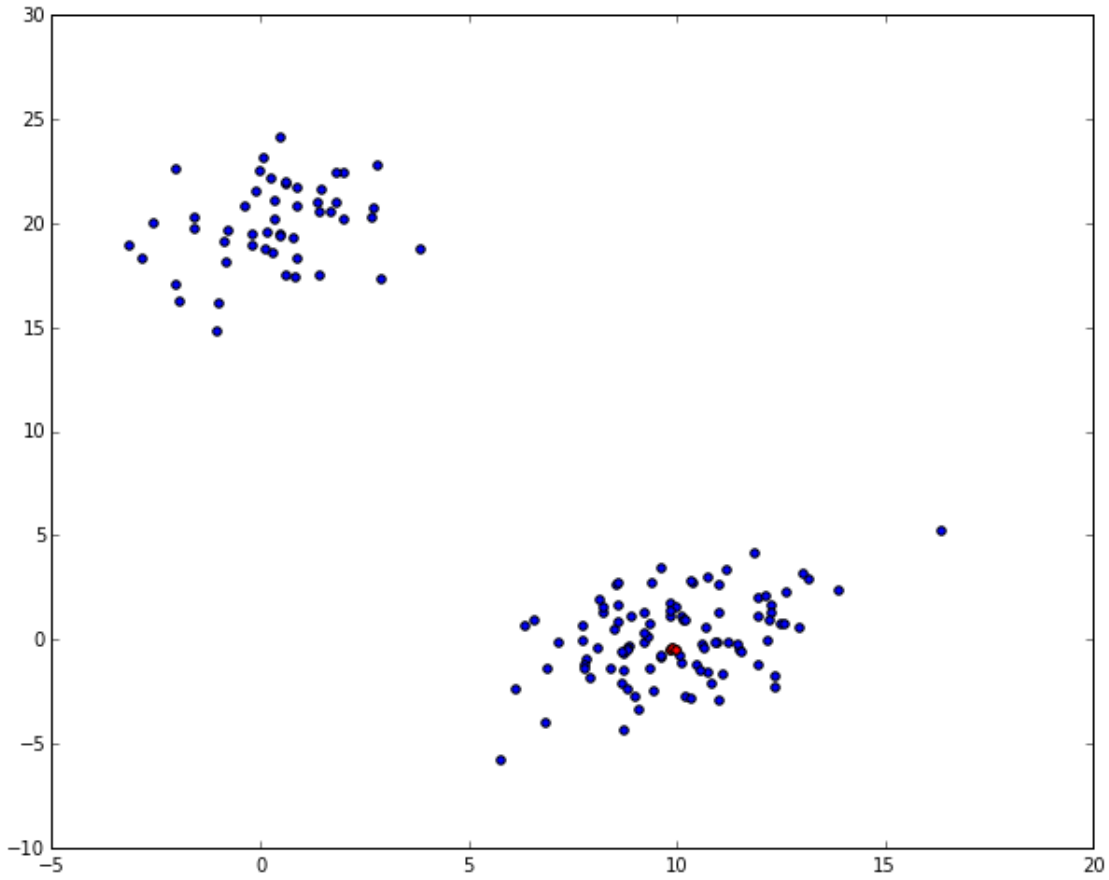
Out[9]:

```
array([[ 9.83913, -0.4873 ],
       [ 9.89349, -0.44152],
       [ 9.97793, -0.56383]])
```

Seems pretty close, but let's plot the points again and highlight them:

In [10]:

```
idxs = [33, 68, 62]
plt.figure(figsize=(10, 8))
plt.scatter(X[:,0], X[:,1])  # plot all points
plt.scatter(X[idxs,0], X[idxs,1], c='r')  # plot interesting points in red again
plt.show()
```

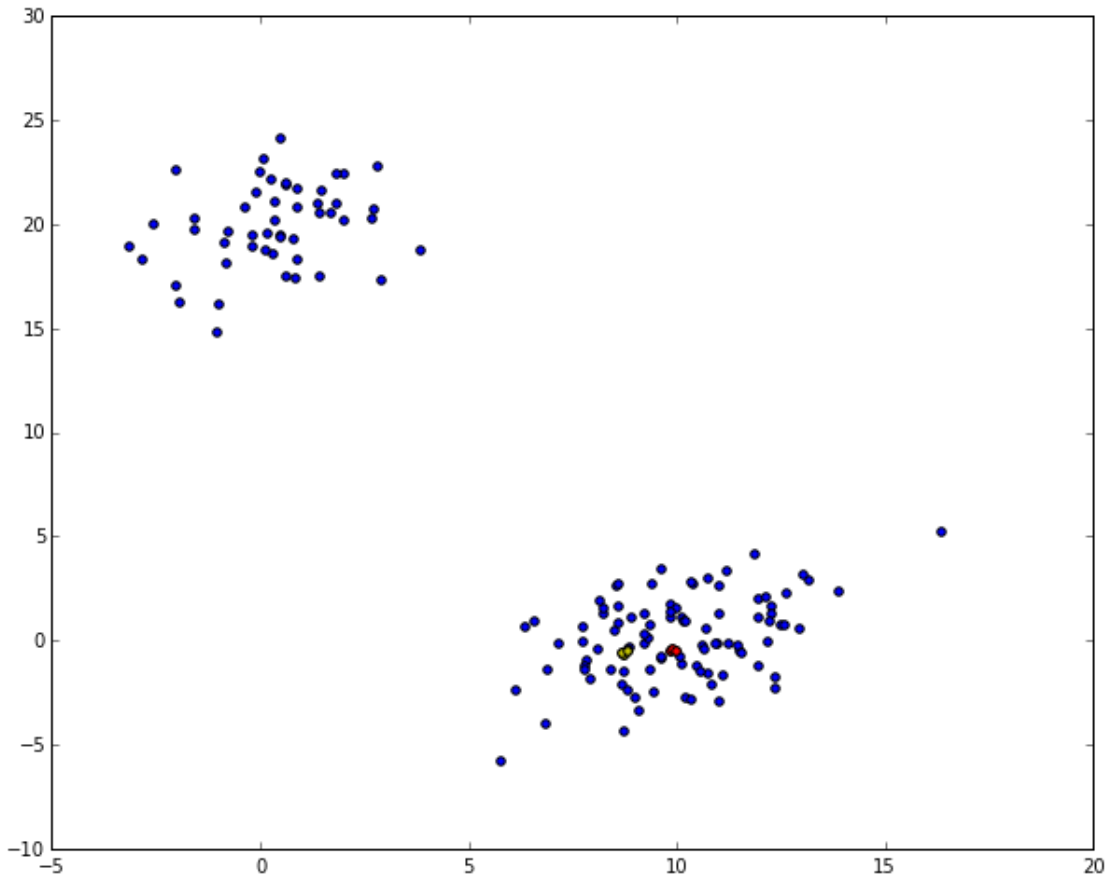


We can see that the 3 red dots are pretty close to each other, which is a good thing.

The same happened in iteration 14 where the alrogithm merged indices 41 to 15 and 69:

In [11]:

```
idxs = [33, 68, 62]
plt.figure(figsize=(10, 8))
plt.scatter(X[:,0], X[:,1])
plt.scatter(X[idxs,0], X[idxs,1], c='r')
idxs = [15, 69, 41]
plt.scatter(X[idxs,0], X[idxs,1], c='y')
plt.show()
```



Showing that the 3 yellow dots are also quite close.

And so on...

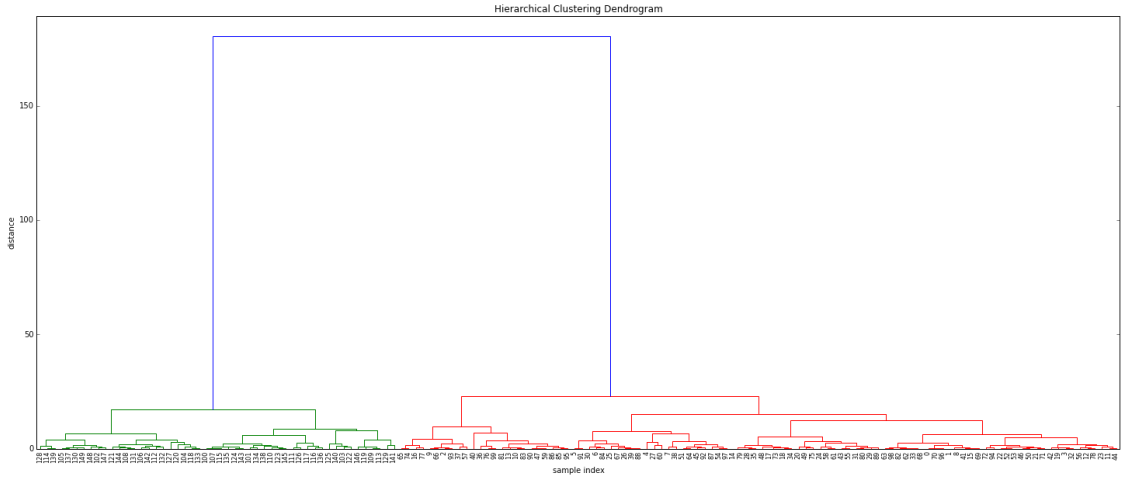
We'll later come back to visualizing this, but now let's have a look at what's called a dendrogram of this hierarchical clustering first:

Plotting a Dendrogram

A dendrogram is a visualization in form of a tree showing the order and distances of merges during the hierarchical clustering.

In [12]:

```
# calculate full dendrogram
plt.figure(figsize=(25, 10))
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('sample index')
plt.ylabel('distance')
dendrogram(
    Z,
    leaf_rotation=90.,  # rotates the x axis labels
    leaf_font_size=8.,  # font size for the x axis labels
)
plt.show()
```



(right click and "View Image" to see full resolution)

If this is the first time you see a dendrogram, it's probably quite confusing, so let's take this apart...

- On the x axis you see labels. If you don't specify anything else they are the indices of your samples in `X`.
- On the y axis you see the distances (of the `'ward'` method in our case).

Starting from each label at the bottom, you can see a vertical line up to a horizontal line. The height of that horizontal line tells you about the distance at which this label was merged into another label or cluster. You can find that other cluster by following the other vertical line down again. If you don't encounter another horizontal line, it was just merged with the other label you reach, otherwise it was merged into another cluster that was formed earlier.

Summarizing:

- horizontal lines are cluster merges
- vertical lines tell you which clusters/labels were part of merge forming that new cluster
- heights of the horizontal lines tell you about the distance that needed to be "bridged" to form the new cluster

You can also see that from distances > 25 up there's a huge jump of the distance to the final merge at a distance of approx. 180. Let's have a look at the distances of the last 4 merges:

```
In [13]:
Z[-4:,2]
Out[13]:
array([  15.11533,   17.11527,   23.12199,  180.27043])
```

Such distance jumps / gaps in the dendrogram are pretty interesting for us. They indicate that something is merged here, that maybe just shouldn't be merged. In other words: maybe the things that were merged here really don't belong to the same cluster, telling us that maybe there's just 2 clusters here.

Looking at indices in the above dendrogram also shows us that the green cluster only has indices >= 100, while the red one only has such < 100. This is a good thing as it shows that the algorithm re-discovered the two classes in our toy example.

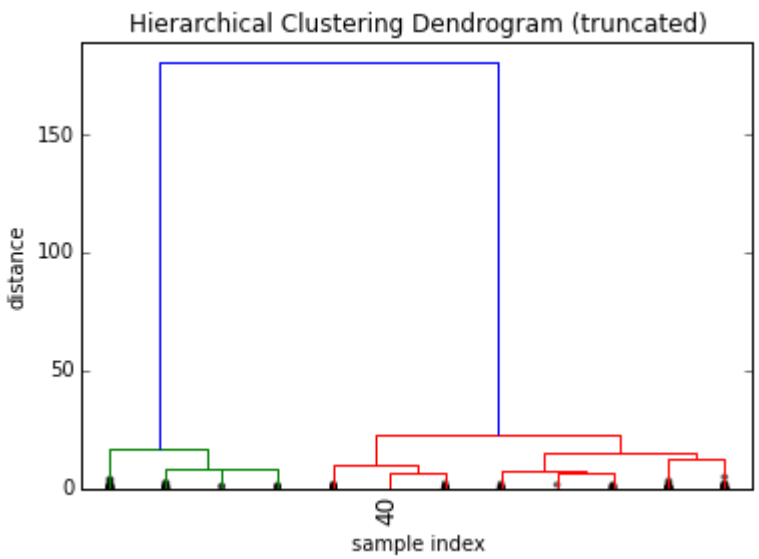
In case you're wondering about where the colors come from, you might want to have a look at the `color_threshold` argument of `dendrogram()`, which as not specified automagically picked a distance cut-off value of 70 % of the final merge and then colored the first clusters below that in individual colors.

Dendrogram Truncation

As you might have noticed, the above is pretty big for 150 samples already and you probably have way more in real scenarios, so let me spend a few seconds on highlighting some other features of the [dendrogram()] function:

In [14]:

```
plt.title('Hierarchical Clustering Dendrogram (truncated)')
plt.xlabel('sample index')
plt.ylabel('distance')
dendrogram(
    Z,
    truncate_mode='lastp',  # show only the last p merged clusters
    p=12,  # show only the last p merged clusters
    show_leaf_counts=False,  # otherwise numbers in brackets are counts
    leaf_rotation=90.,
    leaf_font_size=12.,
    show_contracted=True,  # to get a distribution impression in truncated branches
)
plt.show()
```



The above shows a truncated dendrogram, which only shows the last `p=12` out of our 149 merges.

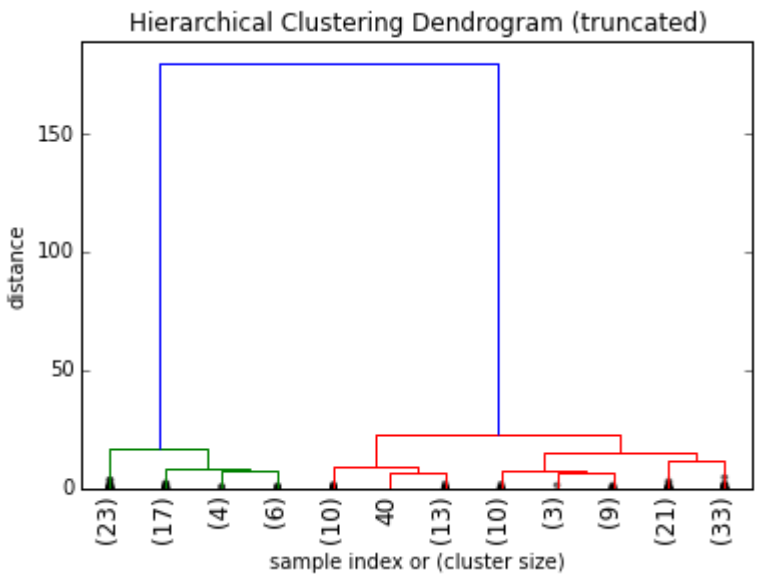
First thing you should notice are that most labels are missing. This is because except for `X[40]` all other samples were already merged into clusters before the last 12 merges.

The parameter `show_contracted` allows us to draw black dots at the heights of those previous cluster merges, so we can still spot gaps even if we don't want to clutter the whole visualization. In our example we can see that the dots are all at pretty small distances when compared to the huge last merge at a distance of 180, telling us that we probably didn't miss much there.

As it's kind of hard to keep track of the cluster sizes just by the dots, `dendrogram()` will by default also print the cluster sizes in brackets `()` if a cluster was truncated:

In [15]:

```
plt.title('Hierarchical Clustering Dendrogram (truncated)')
plt.xlabel('sample index or (cluster size)')
plt.ylabel('distance')
dendrogram(
    Z,
    truncate_mode='lastp',  # show only the last p merged clusters
    p=12,  # show only the last p merged clusters
    leaf_rotation=90.,
    leaf_font_size=12.,
    show_contracted=True,  # to get a distribution impression in truncated branches
)
plt.show()
```



We can now see that the right most cluster already consisted of 33 samples before the last 12 merges.

Eye Candy

Even though this already makes for quite a nice visualization, we can pimp it even more by also annotating the distances inside the dendrogram by using some of the useful return values dendrogram():

In [16]:

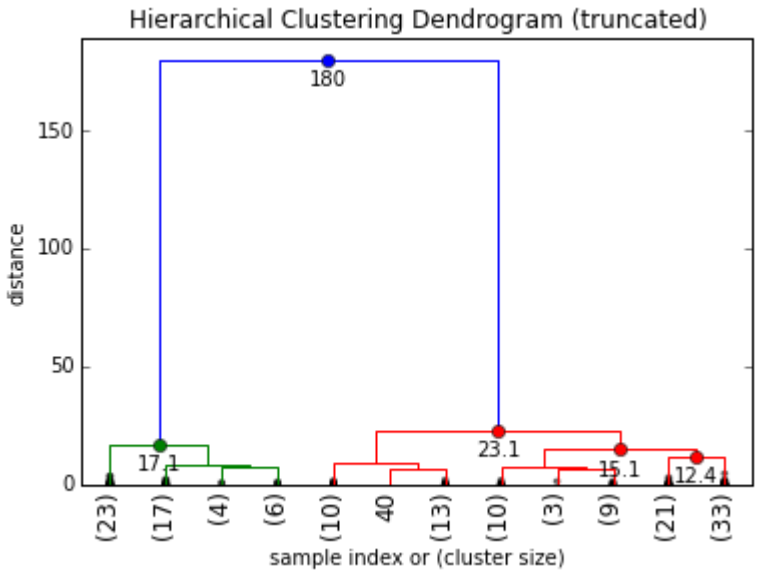
```
def fancy_dendrogram(*args, **kwargs):
    max_d = kwargs.pop('max_d', None)
    if max_d and 'color_threshold' not in kwargs:
        kwargs['color_threshold'] = max_d
    annotate_above = kwargs.pop('annotate_above', 0)

    ddata = dendrogram(*args, **kwargs)

    if not kwargs.get('no_plot', False):
        plt.title('Hierarchical Clustering Dendrogram (truncated)')
        plt.xlabel('sample index or (cluster size)')
        plt.ylabel('distance')
        for i, d, c in zip(ddata['icoord'], ddata['dcoord'], ddata['color_list']):
            x = 0.5 * sum(i[1:3])
            y = d[1]
            if y > annotate_above:
                plt.plot(x, y, 'o', c=c)
                plt.annotate("%.3g" % y, (x, y), xytext=(0, -5),
                             textcoords='offset points',
                             va='top', ha='center')
        if max_d:
            plt.axhline(y=max_d, c='k')
    return ddata
```

In [17]:

```
fancy_dendrogram(
    Z,
    truncate_mode='lastp',
    p=12,
    leaf_rotation=90.,
    leaf_font_size=12.,
    show_contracted=True,
    annotate_above=10,  # useful in small plots so annotations don't overlap
)
plt.show()
```



Selecting a Distance Cut-Off aka Determining the Number of Clusters

As explained above already, a huge jump in distance is typically what we're interested in if we want to argue for a certain number of clusters. If you have the chance to do this manually, i'd always opt for that, as it allows you to gain some insights into your data and to perform some sanity checks on the edge cases. In our case i'd probably just say that our cut-off is 50, as the jump is pretty obvious:

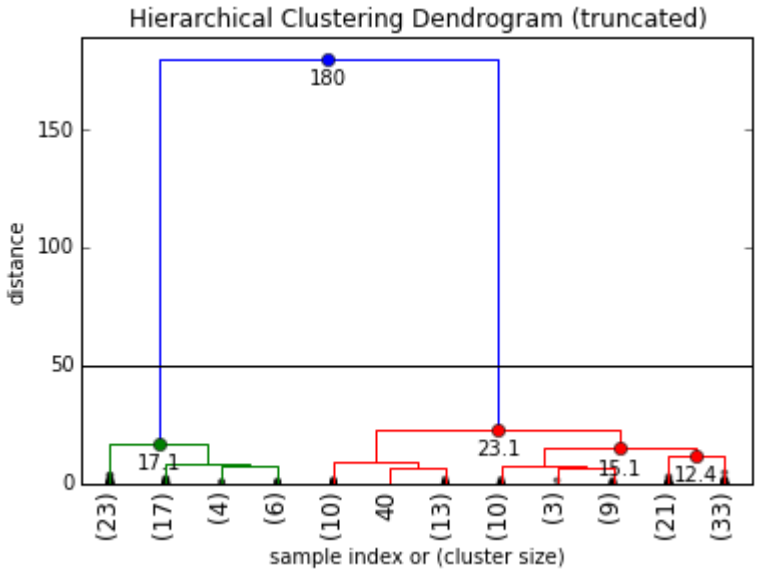
In [18]:

```
# set cut-off to 50
max_d = 50  # max_d as in max_distance
```

Let's visualize this in the dendrogram as a cut-off line:

In [19]:

```
fancy_dendrogram(
    Z,
    truncate_mode='lastp',
    p=12,
    leaf_rotation=90.,
    leaf_font_size=12.,
    show_contracted=True,
    annotate_above=10,
    max_d=max_d,  # plot a horizontal cut-off line
)
plt.show()
```

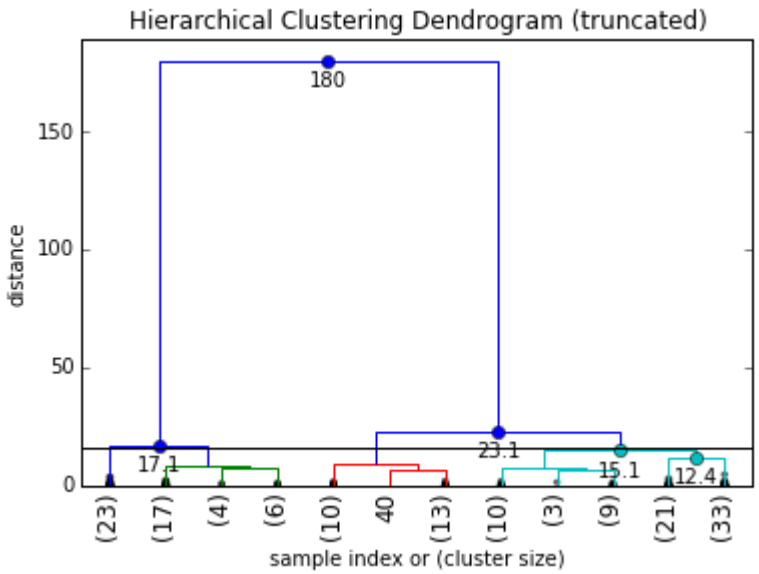


As we can see, we ("surprisingly") have two clusters at this cut-off.

In general for a chosen cut-off value `max_d` you can always simply count the number of intersections with vertical lines of the dendrogram to get the number of formed clusters. Say we choose a cut-off of `max_d = 16`, we'd get 4 final clusters:

In [20]:

```
fancy_dendrogram(
    Z,
    truncate_mode='lastp',
    p=12,
    leaf_rotation=90.,
    leaf_font_size=12.,
    show_contracted=True,
    annotate_above=10,
    max_d=16,
)
plt.show()
```



Automated Cut-Off Selection (or why you shouldn't rely on this)

Now while this manual selection of a cut-off value offers a lot of benefits when it comes to checking for a meaningful clustering and cut-off, there are cases in which you want to automate this.

The problem again is that there is no golden method to pick the number of clusters for all cases (which is why i think the investigative & backtesting manual method is preferable). Wikipedia lists a couple of common methods. Reading this, you should realize how different the approaches and how vague their descriptions are.

I honestly think it's a really bad idea to just use any of those methods, unless you know the data you're working on really really well.

Inconsistency Method

For example, let's have a look at the "inconsistency" method, which seems to be one of the defaults for the `fcluster()` function in scipy.

The question driving the inconsistency method is "what makes a distance jump a jump?". It answers this by comparing each cluster merge's height `h` to the average `avg` and normalizing it by the standard deviation `std` formed over the `depth` previous levels:

$$inconsistency = \frac{h - avg}{std}$$

The following shows a matrix of the `avg, std, count, inconsistency` for each of the last 10 merges of our hierarchical clustering with `depth = 5`

In [21]:

```
from scipy.cluster.hierarchy import inconsistent

depth = 5
incons = inconsistent(Z, depth)
incons[-10:]
```

Out[21]:

```
array([[  1.80875,   2.17062,  10.     ,   2.44277],
       [  2.31732,   2.19649,  16.     ,   2.52742],
       [  2.24512,   2.44225,   9.     ,   2.37659],
       [  2.30462,   2.44191,  21.     ,   2.63875],
       [  2.20673,   2.68378,  17.     ,   2.84582],
       [  1.95309,   2.581  ,  29.     ,   4.05821],
       [  3.46173,   3.53736,  28.     ,   3.29444],
       [  3.15857,   3.54836,  28.     ,   3.93328],
       [  4.9021 ,   5.10302,  28.     ,   3.57042],
       [ 12.122  ,  32.15468,  30.     ,   5.22936]])
```

Now you might be tempted to say "yay, let's just pick 5" as a limit in the inconsistencies, but look at what happens if we set depth to 3 instead:

In [22]:

```
depth = 3
incons = inconsistent(Z, depth)
incons[-10:]
```

Out[22]:

```
array([[  3.63778,   2.55561,   4.     ,   1.35908],
       [  3.89767,   2.57216,   7.     ,   1.54388],
       [  3.05886,   2.66707,   6.     ,   1.87115],
       [  4.92746,   2.7326 ,   7.     ,   1.39822],
       [  4.76943,   3.16277,   6.     ,   1.60456],
       [  5.27288,   3.56605,   7.     ,   2.00627],
       [  8.22057,   4.07583,   7.     ,   1.69162],
       [  7.83287,   4.46681,   7.     ,   2.07808],
       [ 11.38091,   6.2943 ,   7.     ,   1.86535],
       [ 37.25845,  63.31539,   7.     ,   2.25872]])
```

Oups! This should make you realize that the inconsistency values heavily depend on the depth of the tree you calculate the averages over.

Another problem in its calculation is that the previous d levels' heights aren't normally distributed, but expected to increase, so you can't really just treat the current level as an "outlier" of a normal distribution, as it's expected to be bigger.

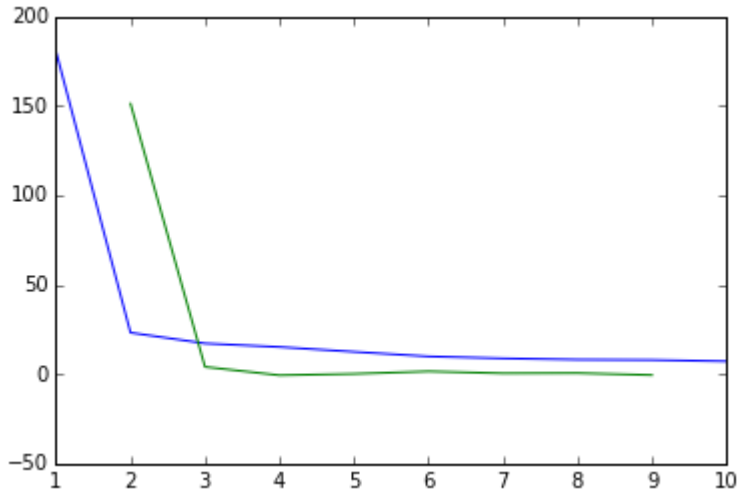
Elbow Method

Another thing you might see out there is a variant of the "elbow method". It tries to find the clustering step where the acceleration of distance growth is the biggest (the "strongest elbow" of the blue line graph below, which is the highest value of the green graph below):

In [23]:

```
last = Z[-10:, 2]
last_rev = last[::-1]
idxs = np.arange(1, len(last) + 1)
plt.plot(idxs, last_rev)

acceleration = np.diff(last, 2)  # 2nd derivative of the distances
acceleration_rev = acceleration[::-1]
plt.plot(idxs[:-2] + 1, acceleration_rev)
plt.show()
k = acceleration_rev.argmax() + 2  # if idx 0 is the max of this we want 2 clusters
print "clusters:", k
```




```
clusters: 2
```

While this works nicely in our simplistic example (the green line takes its maximum for k=2), it's pretty flawed as well.

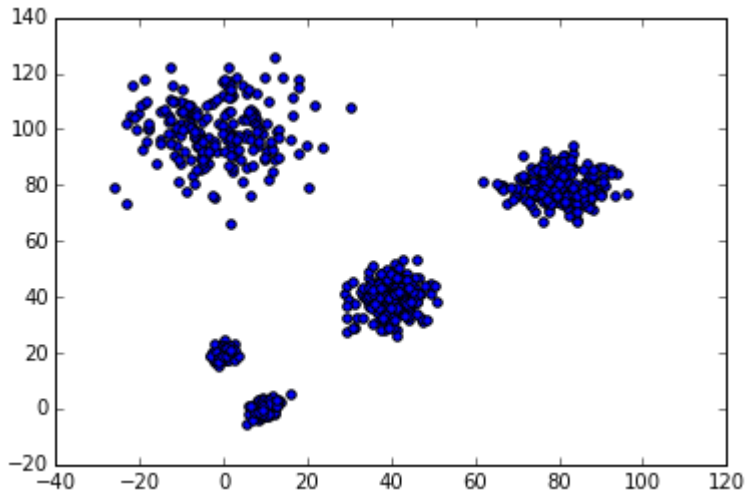
One issue of this method has to do with the way an "elbow" is defined: you need at least a right and a left point, which implies that this method will never be able to tell you that all your data is in one single cluster only.

Another problem with this variant lies in the `np.diff(Z[:, 2], 2)` though. The order of the distances in `Z[:, 2]` isn't properly reflecting the order of merges within one branch of the tree. In other words: there is no guarantee that the distance of `Z[i]` is contained in the branch of `Z[i+1]`. By simply computing the `np.diff(Z[:, 2], 2)` we assume that this doesn't matter and just compare distance jumps from different branches of our merge tree.

If you still don't want to believe this, let's just construct another simplistic example but this time with very different variances in the different clusters:

In [24]:

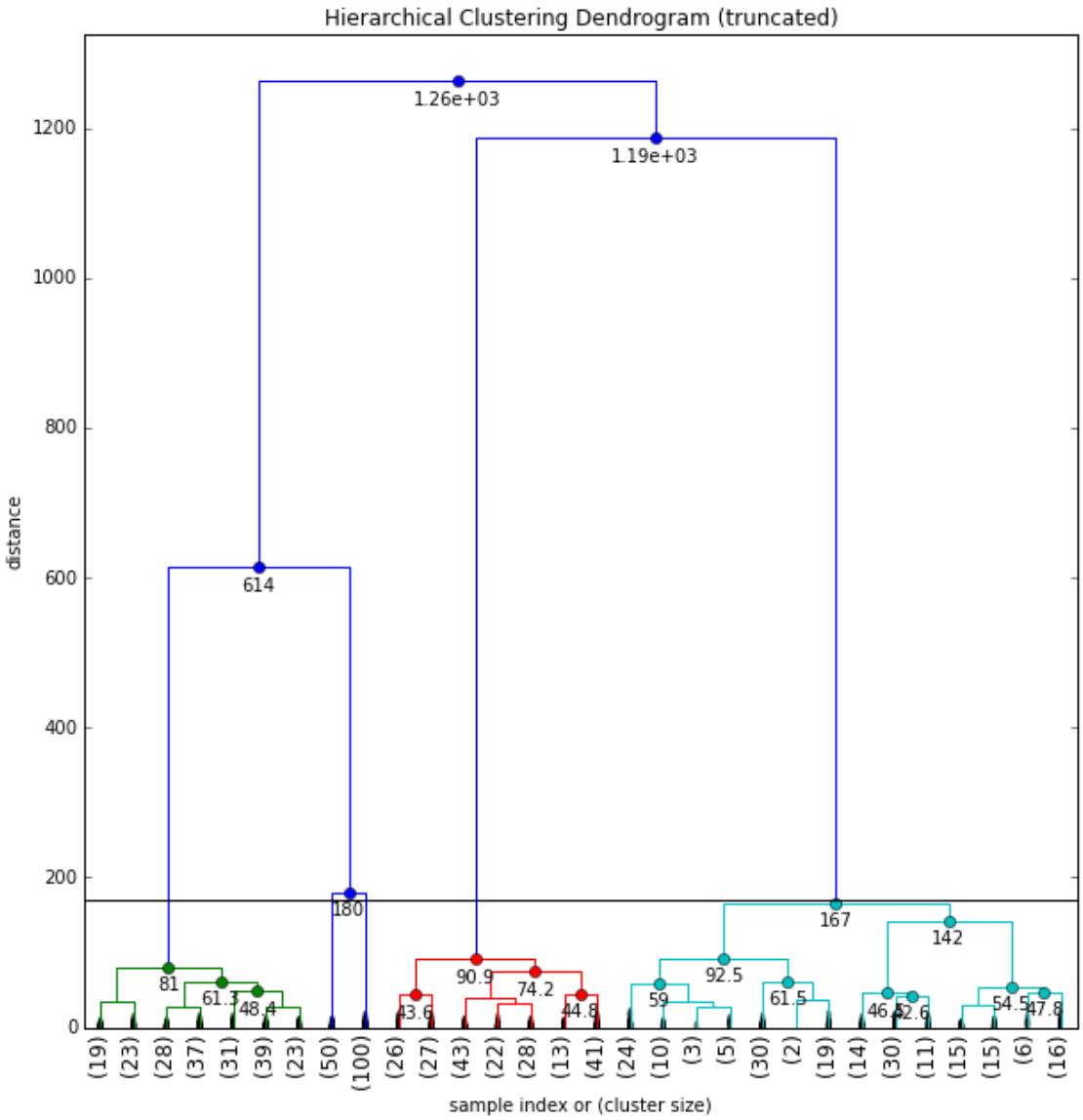
```
c = np.random.multivariate_normal([40, 40], [[20, 1], [1, 30]], size=[200,])
d = np.random.multivariate_normal([80, 80], [[30, 1], [1, 30]], size=[200,])
e = np.random.multivariate_normal([0, 100], [[100, 1], [1, 100]], size=[200,])
X2 = np.concatenate((X, c, d, e),)
plt.scatter(X2[:,0], X2[:,1])
plt.show()
```



As you can see we have 5 clusters now, but they have increasing variances... let's have a look at the dendrogram again and how you can use it to spot the problem:

In [25]:

```
Z2 = linkage(X2, 'ward')
plt.figure(figsize=(10,10))
fancy_dendrogram(
    Z2,
    truncate_mode='lastp',
    p=30,
    leaf_rotation=90.,
    leaf_font_size=12.,
    show_contracted=True,
    annotate_above=40,
    max_d=170,
)
plt.show()
```



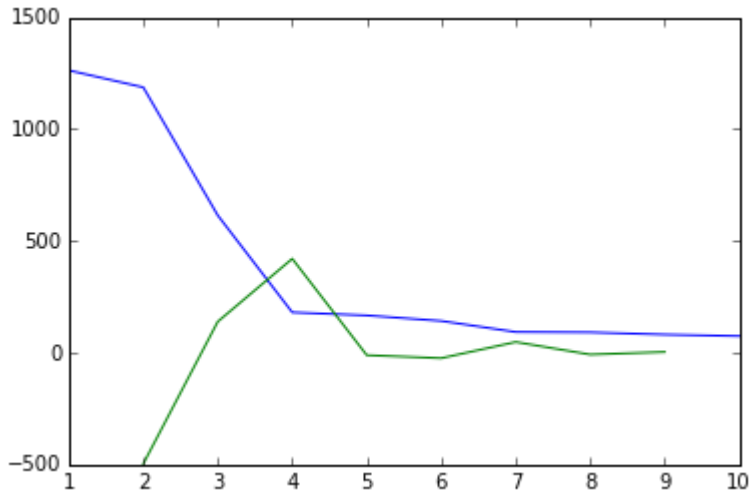
When looking at a dendrogram like this and trying to put a cut-off line somewhere, you should notice the very different distributions of merge distances below that cut-off line. Compare the distribution in the cyan cluster to the red, green or even two blue clusters that have even been truncated away. In the cyan cluster below the cut-off we don't really have any discontinuity of merge distances up to very close to the cut-off line. The two blue clusters on the other hand are each merged below a distance of 25, and have a gap of > 155 to our cut-off line.

The variant of the "elbow" method will incorrectly see the jump from 167 to 180 as minimal and tell us we have 4 clusters:

In [26]:

```
last = Z2[-10:, 2]
last_rev = last[::-1]
idxs = np.arange(1, len(last) + 1)
plt.plot(idxs, last_rev)

acceleration = np.diff(last, 2)  # 2nd derivative of the distances
acceleration_rev = acceleration[::-1]
plt.plot(idxs[:-2] + 1, acceleration_rev)
plt.show()
k = acceleration_rev.argmax() + 2  # if idx 0 is the max of this we want 2 clusters
print "clusters:", k
```




```
clusters: 4
```

The same happens with the inconsistency metric:

In [27]:

```
print inconsistent(Z2, 5)[-10:]
```

```
[[  13.99222   15.56656   30.         3.86585]
 [  16.73941   18.5639    30.         3.45983]
 [  19.05945   20.53211   31.         3.49953]
 [  19.25574   20.82658   29.         3.51907]
 [  21.36116   26.7766    30.         4.50256]
 [  36.58101   37.08602   31.         3.50761]
 [  12.122     32.15468   30.         5.22936]
 [  42.6137   111.38577   31.         5.13038]
 [  81.75199  208.31582   31.         5.30448]
 [ 147.25602  307.95701   31.         3.6215 ]]
```

I hope you can now understand why i'm warning against blindly using any of those methods on a dataset you know nothing about. They can give you some indication, but you should always go back in and check if the results make sense, for example with a dendrogram which is a great tool for that (especially if you have higher dimensional data that you can't simply visualize anymore).

Retrieve the Clusters

Now, let's finally have a look at how to retrieve the clusters, for different ways of determining k. We can use the `fcluster` function.

Knowing max_d:

Let's say we determined the max distance with help of a dendrogram, then we can do the following to get the cluster id for each of our samples:

In [28]:

```
from scipy.cluster.hierarchy import fcluster
max_d = 50
clusters = fcluster(Z, max_d, criterion='distance')
clusters
```

Out[28]:

```
array([2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], dtype=int32)
```

Knowing k:

Another way starting from the dendrogram is to say "i can see i have k=2" clusters. You can then use:

In [29]:

```
k=2
fcluster(Z, k, criterion='maxclust')
```

Out[29]:

```
array([2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], dtype=int32)
```

Using the Inconsistency Method (default):

If you're really sure you want to use the inconsistency method to determine the number of clusters in your dataset, you can use the default `criterion` of `fcluster()` and hope you picked the correct values:

In [30]:

```
from scipy.cluster.hierarchy import fcluster
fcluster(Z, 8, depth=10)
```

Out[30]:

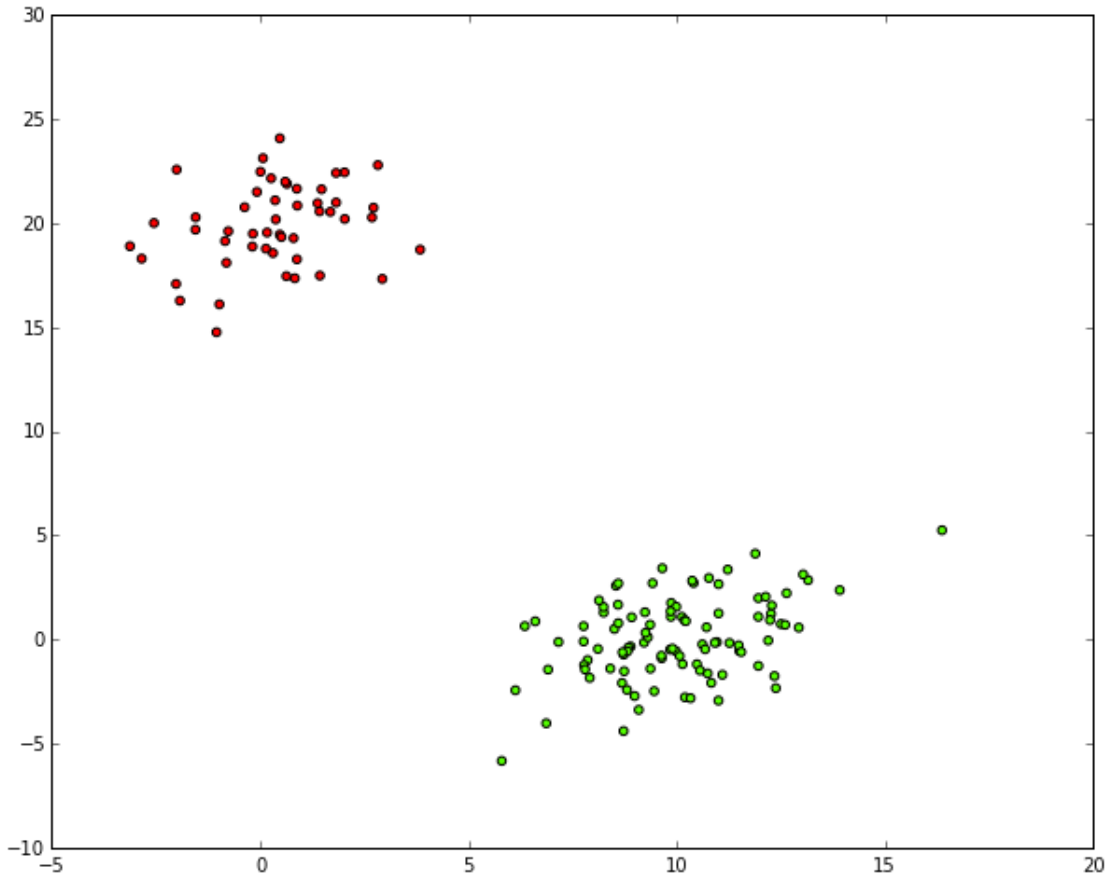
```
array([2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], dtype=int32)
```

Visualizing Your Clusters

If you're lucky enough and your data is very low dimensional, you can actually visualize the resulting clusters very easily:

In [31]:

```
plt.figure(figsize=(10, 8))
plt.scatter(X[:,0], X[:,1], c=clusters, cmap='prism')  # plot points with cluster dependen
plt.show()
```



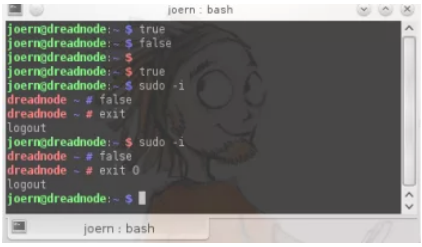
I hope you enjoyed this tutorial. Feedback welcome ;)

Further Reading:

- The scipy hierarchical clustering module: http://docs.scipy.org/doc/scipy/reference/cluster.hierarchy.html
- The scipy distance computation docs: http://docs.scipy.org/doc/scipy/reference/spatial.distance.html
- The scipy hierarchical clustering module docs often refer to the MATLAB docs saying that a certain function is similar to the MATLAB one. Here's their hierarchical clustering tutorial: http://mathworks.com/help/stats/hierarchical-clustering.html

Related

[How to convert hex strings to binary ascii strings in python (incl. 8bit space)](#)
As i come across this again and again: How do you turn a hex string like "c3a4c3b6c3bc" into a nice binary string like this: "11000011 10100100 11000011
2010-09-21
In “Coding”



[Bash prompt indicating return value](#)
2010-08-20
In “Coding”

[Min-Heap in Python](#)
I recently wanted to implement a small event system where events can have different priorities. So for example the event with highest priority (lowest val-
2010-07-19
In “Coding”

This entry was posted in Coding and tagged clustering, code, dendrogram, hierarchical clustering, howto, python, scipy, tutorial on 2015-08-26 [https://joernhees.de/blog/2015/08/26/scipy-hierarchical-clustering-and-dendrogram-tutorial/] .

114 thoughts on “SciPy Hierarchical Clustering and Dendrogram Tutorial”

Pedro Marcelino
2015-10-23 at 11:13

This is great stuff. Thanks for your sharing.

Elaine
2015-10-29 at 03:14

I am doing same as your code, but I don’t get the same for Z[0]. I am getting array([5.20000000e+01, 5.30000000e+01, 4.15105485e-02, 2.00000000e+00]). Do you know why?

Thank you.

joern Post author
2015-10-29 at 15:17

are you sure you set the random number generator seed with random.seed(4711) immediately before creating the data?

Haifeng
2016-02-16 at 10:24

I really like this tutorial! Thanks a lot for the sharing.

David H Lubbers
2018-08-25 at 03:08

Very nice. Exactly what I needed. Just made sense to me start to finish.

Irene
2015-10-29 at 13:51

Thank you so much. There are no enough articles explaining how works SciPy Hierarchical Clustering for non-technical users.

Rob
2015-11-20 at 11:39

Thanks! Great post!

Reem
2015-11-26 at 13:56

Thanks a lot so useful

Lorenzo
2016-01-07 at 17:00

hey thanks for the great post. regarding the last plotting, from my understanding I specified k=3 so that:

clusters = fcluster(Z,3,criterion='maxclust')

Hence clusters is an array with a number specifying to what cluster each value in Z belongs to. However when I then do:

plt.scatter(Z[:,0],Z[:,1], c=clusters)
plt.show()

I get the following error:

ValueError: Color array must be two-dimensional

any help?

Lorenzo
2016-01-08 at 09:13

Ok found the problem, the issue is the size mismatch between the clusters array and the Z arrays.

joern Post author
2016-01-08 at 16:13

yupp, sorry for the late reply... you just confused X and Z... X can be plotted ([cci]plt.scatter(X[:,0],X[:,1], c=clusters)[/cci]) and contains your points, Z the hierarchical cluster information.

John Miller
2016-02-03 at 22:38

I'm confused about the `linkage` method....

The documentation says that it takes as input a “condensed distance matrix”. My confusion is that it seems that in this tutorial we are giving `linkage` the sample data directly (Z = linkage(X, ‘ward’)), but I thought I would have to make a “distance matrix” first (eg with `pdist`).

Can you please clarify how to use `linkage`?

joern Post author
2016-02-04 at 09:52

in a pythonic manner the linkage method can work with both as described in its docs: “A condensed or redundant distance matrix. A condensed distance matrix is a flat array containing the upper triangular of the distance matrix. This is the form that pdist returns. Alternatively, a collection of mm observation vectors in n dimensions may be passed as an m by n array.”
http://docs.scipy.org/doc/scipy/reference/generated/scipy.cluster.hierarchy.linkage.html

---

John Miller
2016-02-04 at 17:46

I see, thank you!

So, the following two options would produce identical output:

```
# 1
Z = linkage(X, method='ward')
# 2
Y = pdist(X, method='ward')
Z = linkage(Y)
```

Kevin Eger
2016-02-11 at 19:00

This is an awesome post! Really informative and exactly the intro I needed. How would the technique change if you were clustering 1D data? ie: if rather than having 2D data points, you instead had a calculated distance between a point and every other point (in an nxn matrix where n: point)?

Kevin Eger
2016-02-11 at 19:09

I guess to rephrase my question, I'll describe what the matrix X would look like:

X.shape == (n, m)
-n: users (samples)
– users 0 – n-1
-m: # of comments (features)
– # of comments: integer value

joern Post author
2016-02-15 at 15:25

i used 2D examples (so m == 2) as they're easiest to visualize, but everything above works for arbitrary m... especially for m == 1 as in your case. You'd have X.shape == (n, 1) ...

Saurabh Goyal
2016-03-07 at 08:36

Thank you for this amazing tutorial.
I wanted to ask if there is anyway to find out the features which is common for a particular cluster. For example in document classification i would like to know which words are responsible for the formation of a cluster (common keyword for a cluster).

joern Post author
2016-03-09 at 15:09

I think there isn't really "the one answer" for the general case, as finding common features in a cluster only works if your distance function is "decompose-able". The clusters are formed based on distances between whole samples (so over all features).

Now in the document domain a typical distance is the cosine distance on tf-idf word vectors over documents, so you are lucky as that's just the dot product (which is just the sum over the multiplication of all individual features (so the word counts) and can be decomposed). I'd probably approach this by forming "clus-

ter documents". This means that you use the ideas of tf-idf and just apply them to the clusters. Add all the document vectors (so all the word occurrences within a cluster) to get the clusters' term frequencies (tf). Afterwards use them to form an "inverse cluster frequency" (icf) (equivalent to the "inverse document frequency" (idf)), with which you can build the tf-icf. Then per cluster just select the term with the top tf-icf and you should have your label.

That's obviously only one solution and there's a whole lot of research in the field of document analysis about Cluster labeling (https://en.wikipedia.org/wiki/Cluster_labeling).

O.rka
2016-04-09 at 19:12

Thanks for the tutorial this definitely helped get me started. I'm confused on the input Z of the linkage function. I have a similarity matrix, convert it to a dissimilarity matrix, then do I give linkage the np.triu, np.tril, or the square matrix? I posted my question on stackoverflow for figures and examples: http://stackoverflow.-com/questions/36520043/triangle-vs-square-distance-matrix-for-hierarchical-clustering-python

---

Mary
2016-04-28 at 17:19

I keep getting "ValueError: negative dimensions are not allowed" when calling linkage(X, 'ward'),
where X is an ndarray of shape (54000, 100) and represents 54000 observations in a 100 feature space.
What gives?

joern Post author
2016-04-28 at 17:25

what does [cci]print X.shape[/cci] tell you?

Mary
2016-04-28 at 17:31

It says (54000L, 100L) Not sure if the L would have something to do with this.

joern Post author
2016-04-28 at 17:38

54000 by far isn't long, so seems you're doing something weird there...
can you manually invoke the pdist method and pass it to linkage like this?

```
from scipy.spatial.distance import pdist
Y = pdist(X)
Z = linkage(X, 'ward')
```

Or maybe just re-create the array and check if it still has a weird "long" shape?

Mary
2016-04-28 at 19:06

I did:

```
Y = pdist(X)
Z = linkage(Y, 'ward')
```

and got:
ValueError: Valid methods when the raw observations are omitted are 'single', 'complete', 'weighted', and 'average'.

Then I just replaced 'ward' by 'average' and got the initial ValueError of negatives

I initialize X as:

```
X = np.zeros((len(docs), len(ct)), dtype=np.int32)
```

I tried the simple int too and np.int16 for the dtype parameter and I still get the L when calling shape. I used your example and got L too.

```
>>> a = np.random.multivariate_normal([10, 0], [[3, 1], [1, 4]], size=[100,])
>>> b = np.random.multivariate_normal([0, 20], [[3, 1], [1, 4]], size=[50,])
>>> X = np.concatenate((a, b),)
>>> X.shape
(150L, 2L)
```

I have two versions of python installed on my system and I'm running this on python 2.7.11 I don't know what else to say except that I'm going back to R.

Mary
2016-04-28 at 19:12

Nvm. I tried the same initialization line in python 3 and it works fine. 😐

Mithun
2016-05-11 at 13:27

Awesome tutorial, mate! You made the life a lot easier!

joern Post author
2016-05-11 at 13:32

😉

Arnold
2016-05-19 at 17:27

Hi, how can i get all sample indices within the cluster, how can you get it from scipy.cluster.hierarchy.dendrogram? i plan to create a dictionary: key = cluster_index, value = list of sample_index

joern Post author
2016-05-19 at 18:12

```
from collections import defaultdict
cluster_dict = defaultdict(list)
for i, c in enumerate(clusters):
    cluster_dict[c].append(i)
print(cluster_dict)
```

Arnold
2016-05-19 at 22:21

Hi,

thanks for the info.

i have also a question to the normalization of the data:
my feature matrix has the following format: X : array-like, shape = [n_samples, n_features]
how can i scale to zero mean and variance of 1? Im not sure if http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.scale.html is what i look for...

how does the dendrogram coloring work, it seems not each cluster gets its own color in the dendrogram?
ploting the dendrogram: ddata = dendrogram(hc, color_threshold=cut_off, leaf_rotation=90.)
plt.axhline(y=cut_off, linewidth=4, color='r', linestyle='–') plt.show() here is the dendrogram:
http://s32.postimg.org/nhjilnedh/Screen_Shot_2016_05_19_at_6_06_27_PM.png

from the dendogram i know that cluster 1 which consists of sample 1,2,3 ... how can i figure out which feature mostly represents cluster 1? is there anything else i tell about cluster 1?

joern Post author
2016-05-20 at 11:18

yupp, you can use that scaling function to scale...

coloring: for each cluster beneath the threshold it picks a different color

feature representation of the clusters depends on the distance function you use:
https://joernhees.de/blog/2015/08/26/scipy-hierarchical-clustering-and-dendrogram-tutorial/#comment-3648

Arnold
2016-06-07 at 22:01

Hi,

for the following image you said there are 5 clusters.
see plot after “As you can see we have 5 clusters now, but they have increasing variances... let’s have a look at the dendrogram again and how you can use it to spot the problem:”

but the image shows only 4 different color. why is that?

how can you extract the most important feature/s forming a cluster? assume i have 30 features.

Thanks,
Arnold

joern Post author
2016-06-08 at 09:13

that has to do with the color threshold parameter in the [cci]fancy_dendrogram[/cci]. If you look closely you will see that the vertical lines crossing the limit line are always blue... Just that in the case of the very small clusters they are completely collapsed in the shown dendrogram so that you can only see them as blue lines.

wrt. the features have a look at the answers above.

Nghia
2016-06-11 at 17:00

Thank you for the amazing tutorial.
I have a question regards to the max_d values in fcluster function. How can we define acceptable max_d's values?

joern Post author
2016-06-21 at 16:02

one of the main points in the post is to show that there isn't a single definition of max_d that makes sense in all cases… that's what you need the dendrogram for: to look at it and understand what different cut-offs will do to your data.

In general you should look for "jumps" in the distances, but be suspicious about imbalances of the distributions (a mixture of very tight clusters and very wide ones). A single selection of max_d might not be enough in case your cluster distributions are skewed. You will see this in the dendrogram when you can't simply put a horizontal line to cut through all the huge jumps of all merges.

Jo
2016-06-21 at 15:07

Let me first thank you for this great tutorial !
However something still bothers me, I'll try to make it simple.
I have 5 arrays: 0, 1, 2, 3, 4
The corresponding linkage output is as follow:
[[0. 3. 0.03376334 2.]
[2. 1. 0.03924361 2.]
[4. 7. 0.04020413 3.]]
How am I supposed to know which of first cluster ("0" & "3") or second cluster ("2" & "1") is the one named "7", to which the array "4" links ?
Thanks in advance for your consideration,
Jo

joern Post author
2016-06-21 at 15:51

from the post: All indices idx >= len(X) actually refer to the cluster formed in Z[idx – len(X)]

your example seems constructed (and wrong), as a clustering of 5 points requires 4 pairwise merges… wrong because the last merge seems to be self-referring.

Jo
2016-06-21 at 16:35

I did omit he last line of the linkage output… my bad.
Thank you for your answer, it was indeed already explain in your post… I'll read better next time !
Thanks again for this clear tuto !

Berny
2016-06-30 at 13:04

Hi, thanks for this tutorial.
I want to ask you if it is possible to use a function define by me to calculate the distance or i must use the distance defined by the scipy.
Thanks for your answer

joern Post author
2016-08-12 at 18:36

you can use self defined distance functions

Josh
2016-07-02 at 01:54

How do you use `link_color_func` in `dendrogram` ? I have a dictionary where `{key=node/leaf : value=hex_color}` ?

O.rka
2016-07-03 at 23:52

Can you explain what's happening in:
"`
for i, d, c in zip(ddata['icoord'], ddata['dcoord'], ddata['color_list']):
x = 0.5 * sum(i[1:3])
y = d[1]
"`

joern Post author
2016-08-12 at 18:44

hehe, yeah, that's a "hack" to draw the annotations for the heights.

x is the middle between the two connected clusters (so the middle between the vertical lines).

y is the height of the bridge.

If we're above a certain height, we'll plot an additional point in the cluster's color and write the height value below it.

Will
2016-08-12 at 02:12

Thanks for the post, very helpful.
In case it's useful to others reading here: fcluster inexplicably returns an array whose minimum entry is 1, not 0. Beware when using it to index numpy arrays.

Alejandro
2016-10-07 at 21:13

great, great tutorial!

greetings from BA, Argentina

joern Post author

2016-10-10 at 17:58

thanks, glad it helped

rony

2016-10-10 at 08:56

Thank you.. awesome post..!!

joern Post author

2016-10-10 at 18:00

thanks, and i'm happy to see that i'm not the only one who uses spam-tracking emails 😉

Mark G

2016-10-14 at 18:37

Very well written and insightful, thank you Joern

Kyle

2016-10-18 at 20:59

Thank you so much for this, it was a really excellent tutorial!

Nutchaya P.

2016-10-24 at 15:35

Thank you so much, your tutorial is very helpful.
I have a question. For the dendrogram on the x axis, how can I label it with the values of the samples instead of indices ?

Meg
2016-10-28 at 06:55

Thank you very much. This is truly great! I have a question though. Do you know of a way to introduce new samples to the process after the clustering is finished? Just like when we expand an SOM. What I want to do is to create my clusters with a training set and test it with a testing set. Is that possible?

kerr h
2016-12-29 at 09:03

great post. one of the reasons people use python so much is the quality of the online material and examples available, and your tutorial on clustering is a perfect example.

joern Post author
2016-12-30 at 19:08

🙂 thanks

Lorien
2017-01-12 at 23:02

I'm using Python, numpy and scipy to do some hierarchical clustering on the output of a topic model I created for text analysis.

I applied my testcorpus to the ldamodel so it became a bag-of-words representation. Then I turned it into a matrix. Now I want to use scipy to make a linkage matrix of my matrix. But it gives the Value Error: setting an array element with a sequence. I guess that this is because only equally shaped arrays can be clustered. And my matrix has a difference in lengths between the lists inside the list of lists. I just don't now how to solve this. Here is a little part of the code. I don't know if it is helpful. I just really hope someone can help me.

```
import numpy as np
X = np.array(corpus)
from matplotlib import pyplot as plt
from scipy.cluster.hierarchy import dendrogram, linkage
Z = linkage(X, 'cosine')
```

joern Post author
2017-01-16 at 10:13

hmm, not knowing how your `corpus` looks like, i can only guess... my best guess is that you're dealing with a list of some kind of LDA dicts: `[{dim1: score1, dim2: score2,...},...]`. As you indicate scipy doesn't really understand this when you do `np.array(corpus)`, so you need to transform that list of feature-mappings into a list of feature-vectors, either manually or with sklearn.feature_extraction.DictVectorizer.fit_transform for example.

Samuel Nde
2018-08-31 at 19:14

@Lorien, I ran into the same problem as you and I figured out that the problem arises because you are supplying a sparse matrix to the linkage function.
Type `type(yourMatrix)` and you will get `scipy.sparse.csr.csr_matrix`
Convert your matrix to a `numpy array` before applying the linkage function.
You can convert your matrix to an array by doing `yourMatrix.toarray()`. So your Z should be defined as:
Z = linkage(yourMatrix.toarray(), 'yourDistanceFunction')
I hope this helps you. Thanks.

AlexGhiti
2017-01-20 at 13:41

Thank you very much for this tutorial, it really helps 🙂

Ernesto
2017-01-25 at 11:28

Only one word to describe your work: Excellent

joern Post author
2017-01-25 at 11:40

thanks 😉

baba
2017-01-27 at 21:36

this is an excellent tutorial, thanks!! Is there an easy way to make this an online algorithm? i.e. if we have new data points, you can merge them into existing clusters?

joern Post author
2017-02-07 at 17:21

thanks. not really (or at least i don't see any). The reason is that a single added data point could bridge points (low level clusters) that are otherwise merged very late in the process. The whole tree would look different in that case.

Joshan
2017-02-01 at 07:04

Tanks for your post.. Helped me a lot.. have any idea about datafiled potential function in clustering techniques.

joern Post author
2017-02-07 at 17:24

thanks. i don't understand the question though, sorry...

shai
2017-02-15 at 12:35

hello, the code for retrieving k clusters does not word, gives an error:
raise ValueError("Invalid RGBA argument: {!r}".format(orig_c))
ValueError: Invalid RGBA argument: 1

any ideas why?

bp
2017-02-18 at 17:05

Hi,

I am new to data analysis and Python in itself. I was looking at hierarchical clustering and chanced on your tutorial. While your tutorial is pretty easy to follow (thank you!), I am confused if I can use it in my use case. I have a complete weighted undirected graph and I need to find clusters in that graph. I am confused how I can draw a dendrogram to visualize my graph such that the y axis ranges from the min of all edge weights to the max of all edge weights. Any help is highly appreciated. Thank you.

joern Post author
2017-02-19 at 14:47

If you have a fully connected graph and just want to use the edge weights as distance directly, you can actually pass a square distance matrix into the linkage function: https://docs.scipy.org/doc/scipy/reference/generated/scipy.cluster.hierarchy.linkage.html
In your case that would just be the weighted adjacency matrix.

bp
2017-02-19 at 15:13

Nevermind, figured it out. Your tutorial is very helpful – Thank you!

Ethan
2017-02-21 at 18:17

Hi, thank you very much for this tutorial it is very useful.

I did not understand how the inconsistency values are used to identify optimal # of clusters. Can you please give more explanation about it.

Thanks

joern Post author
2017-03-30 at 13:40

Hi, uhm the point of my explanations around the inconsistency method are that it's not very reliable. In theory one is interested in outliers of the inconsistency score, but as shown it's somewhat arbitrary. So i'd suggest to look at the inconsistency scores and only take its outliers as indicators for a reasonable "K". I'd always suggest to afterwards go back to the dendrogram and check if it makes sense.

Lucy
2017-03-22 at 19:48

Hi Joern and thank you for this great tutorial. I helped me a lot!

Someone has posted about getting a sample indices within the cluster and stored into a dictionary. I did the solution that you recommended in which each cluster list contains the size and the value, i.e. sample index.

My question is, instead of returning the sample index, what can I do to return the data id itself? In prior to the clustering, I have tagged my data so that each data has a unique id.

The reason for this is because I want to do an analysis that involves identification and measuring the amount of data from the same class that were clustered in the same cluster.

Thanks in advance!
Lucy.

joern Post author
2017-03-30 at 13:57

hi, sorry for the late reply...

I'm not sure i fully understand the question. If all you want to do is get from the idx back to your data, you should be able to do `X[idx]`. If there's further information that is not in `X` connected to that data row, you can always have an external other dictionary to do the lookup!?!

Todd Wight
2017-03-24 at 22:27

This was awesome. Thanks!

Pingback: Hierarchical Clustering | Just My Notes

Federico
2017-03-27 at 15:01

I'm here to investigate how the third column of the dendrogram (distances) is computed. I'm currently using method='single', metric='euclidean' and I don't understand why in the distance column of the dendrogram all values are different from the ones provided in the 2d array of observation vectors. For example, in the first iteration when we merge two clusters of size 1 each, I would expect the distance to be somewhere in the 2d array of observation vectors.

You stated the following:

"No matter what method and metric you pick, the linkage() function will use that method and metric (i.e. cophenet) to calculate the distances of the clusters (starting with your n individual samples (aka data points) as singleton clusters)) and in each iteration will merge the two clusters which have the smallest distance according the selected method and metric."

However, in the documentation for scipy.cluster.hierarchy.linkage, cophenet is never mentioned. In other words, I don't understand why cophenet should bypass method and metric. Could you please support the previous statement?

joern Post author
2017-03-30 at 19:20

yes, i'd also expect the distance to appear in the linkage results. Without a code example i can only guess that you did not calculate the condensed `pdist`, but instead passed `X` into `linkage`. In that case scipy will implicitly calculate the distances. In other words: you won't find rows of `X` in `Z`, but you should find `pdist(X)` rows in `Z`, especially for `method='single'`.

Wrt. the second part, i don't know where you got that quote from. I never made that statement about cophenet. Cophenet is neither metric nor method. Obviously, if you're calculating cophenet on `Z, pdist(X)`, you should pass the corresponding metric into `pdist`: `pdist(X, metric='...')`.

Praful K
2017-03-29 at 22:13

This is an awesome post ! Impressive clarity of thought and buildup. Thank you.

Harry Baker
2017-04-18 at 22:22

Hey, this was an awesome explanation.

I need a little bit of help going further with my clustering. I want to do some labeling of the cluster's I've found, so I'm trying to figure out how to grab every object that appears in each of the final N truncated clusters. I'm clustering documents based on topic model distributions, so I want to know which topics describe each of the clusters that I found.

Thanks!

Abhishek
2017-04-26 at 15:33

Thanks Joern I understood Y axis of the dendrogram from your blog. Which was crucial to differentiate clustring using WARD and AVERAGE.

WARD has 80 as the range of distance (Y axis)
AVERAGE has only 14 as the distance range (Y axis)

By the above details, I concluded that WARD is the better distance measure than AVERAGE.

Please let me know if my hypothesis is appropriate.

Shamiek Mangipudi
2017-04-28 at 19:58

Fantastic tutorial with tremendous insight. Really loved the flow and the ease with which the author presented the content. This should deserve a gold on ELIF.

joern Post author
2017-05-08 at 18:13

😉 thanks

Liron
2017-06-21 at 09:55

This is unbelievable. Clear, well-explained, interactive. Thanks!

miinna
2017-07-26 at 14:43

Hi is it possible to find out all the clustering nodes in a pair without getting merged(what are getting merged after one iteration). For example, I have 2 cluster {1,2}, and {3,2} . the linkage methods are showing [{1,2}], [{3,4}] here 4 is the merged number of {1,2} cluster.am trying to show like [{1,2}],[{3,(1,2)}]
can anybody help...!

Pingback: Baby Steps – Ground Zero

Ram B
2017-08-26 at 08:13

Great Tutorial! I really appreciate you taking the time to write this piece!!

shanhe
2017-08-26 at 10:09

great works! thanks for a lot

Fouad
2017-10-03 at 17:49

Your tutorial is really clear and helpful, thanks a lot !

hewgreen
2017-10-10 at 09:36

Class tutorial pal. Much appreciated.

Thiago
2017-12-13 at 01:22

Thanks so much. It is really very helpful, specially because you discuss on comments, not only the post. You give support to us.

So, if I already have the distance matrix between my data, I have to pass the matrix in triangular shape, otherwise it will be interpreted as the dataset?

In my case, I have pseudo-distance, since diagonal is not zero, but constant equal to 0.5. Should I keep it in the triangular matrix or only the upper part?

Regards.

Navein
2017-12-30 at 10:33

Thanks for your tutorial on this. I was wondering, is it possible to use hierarchical clustering to cluster results from minhash LSH? I looked around and could not find any tutorials or codes on it, but I have read about it in some papers online.

joern Post author
2018-01-22 at 16:01

hmm, tricky question 😉

Depending on your point of view, locality-sensitive hashing can be seen as an alternative to clustering (LSH is already meant to put things that are similar close together in a lower dim vector space), a dimensionality reduction technique or a way to quickly retrieve candidates that are hopefully close together…

So when you ask how to do hierarchical clustering on results from LSH, you could either just apply hierarchical clustering on the lower dim LSH vector space representation of each item, or you could use LSH to quickly retrieve candidates. In the former case, your vector space will very likely have weird properties and will require attention to the selection of your clustering variant and the used distance function. In the latter case, i'd suggest to try and get the original vector space representation (pre-LSH) and then cluster the candidates in that vector space, which is probably much easier to "interpret".

Paco
2018-01-19 at 09:13

Hi Joern and thanks for a nice tutorial.

Now, there comes the question. Is this applicable to time series? My data matrix consists of different daily temperature time series for different locations on each column. I would like to build clusters so I can region-

ally group locations based on temperature time series. Do you think your example could be applied for my problem?

Thanks in advance and best regards

joern Post author
2018-01-22 at 15:37

thanks 🙂

hmmm, not sure if i understand the question. I'd say you that as soon as you have some vector-space representation of your data, you can use clustering techniques. The main difficulty however is finding a meaningful distance measure for your specific use-case.

In your case, the combination of geo-locs, time series and temperatures will most likely not be euclidean 😉 So you'll want to define your own distance measure by asking yourself something like: "What's the distance of two datapoints that are 1km apart and have the same temperature but are recorded 4 months apart?" Vary all combinations of the dimensions and think about how you'd like your "distance" to change. Also be aware that many clustering variants will require certain properties of your defined "distance" like (symmetry, reflexivity, transitivity, triangle inequality...).

Leonardo
2018-06-22 at 15:12

Hi,
For time series clustering and classification I have used a different metric for measures similarities between series, this is called Dynamic Time Warping. You can find a very good explanation and codes in python in http://nbviewer.jupyter.org/github/alexminnaar/time-series-classification-and-clustering/blob/master/Time%20Series%20Classification%20and%20Clustering.ipynb

My question is if I could apply the Hierarchical Clustering using DTW for measure distance between samples.

Michael Rose
2018-01-26 at 01:18

The link in elbow plot named "out there" leads to a random placeholder website.

joern Post author
2018-01-26 at 10:38

thanks, updated 🙂

Alfredo
2018-02-20 at 10:42

Thank you very much for this great, useful tutorial !

Alexandre Larrain
2018-03-28 at 13:03

Thank you very much Jörn, this is very clear and awesome

Pingback: Online reviews: keyword clusters - Data Science at Reputation.com

saurabh chamotra
2018-04-04 at 13:45

Thank you for such an informative blog.I would be thankful to you if you can help me in a problem while i tried your examples on my system.Actually the problem occurred when i tried to draw the dendrogarm i received following error message

IndexError: only integers, slices (`:`), ellipsis (`...`), numpy.newaxis (`None`) and integer or boolean arrays are valid indices

although i used came code provided in your example but somehow i received above mentioned error message

i am using python 2.7 with
numpy 1.12.0b1

scipy ‘0.9.0’
i would be really thankful to you for any support

Cuong
2018-04-04 at 15:52

Thank you for great tutorial!

Jon
2018-04-09 at 15:25

I’m doing my first steps on cluster analysis and this really helped, thank you!
Quick question:
– Is there a simple and comprehensive way to get the data points of each cluster after the initial linkage? I want to get those data points to apply custom criteria and select suitable clusters. Indices in the hierarchical matrix can be extremely complicated, with multiple levels. The “fcluster” function actually selects the final clusters, which is something I don’t want. Have spent some time working on a custom function for this, but I get tangled in infinite loops!

In other words, if I have the points 1,2,3 and I get the following clusters from linkage, (1,2), ((1,2),3), how do I retrieve the actual data points?

mishmeshim
2018-05-03 at 02:36

This was extremely helpful and exactly what I needed! The Scipy documentation on this is really lacking.

Rodrigo
2018-05-03 at 22:00

FAB tutorial

Christos
2018-05-10 at 14:12

Hello,

Very useful article indeed.

In terms of getting k clusters out of linkage, is it possible to use cut_tree and define and get explicitly k clusters (e.g. cutree = cut_tree(Z, n_clusters=10)). If not, is there a way to get explicitly k clusters?

Thank you very much for your time

SangHoon
2018-05-17 at 02:47

Thanks for your tutorial,
My english is not good please understand, but I have question.
I want to hierarchical clustering of Climate data(gridded data)
Dataset: Time(daily: 1970.1.1-2000.12.31), latitude(1degree interval), longitude (1 degree interval).
I expect that “The variation of Pearson correlation coefficients between the two neighbours in group clusters”

Figure 6 in https://www.sciencedirect.com/science/article/pii/S0022169414008385

Leonardo
2018-06-21 at 16:57

Hello, thanks for this tuturial

I have a question. Can I have use my own metric for distance calculations?

Mostafa
2018-08-06 at 17:42

great tutorial, really it is the best tutorial i’ve ever seen on that topic

Andor Samuel Kesselman
2018-09-19 at 18:30

As a potential extension of this, I believe persistent homology would be another appropriate method for determining the correct number of clusters that most likely and accurately describe the topology of the data with respect to the linkages and optimal clustering formations.

https://en.wikipedia.org/wiki/Persistent_homology

You're right that there's different strokes for different folks when it comes to correctly computing the number of clusters, but I probably will differ in my aversion to automatic methods when it comes to clustering analysis. I think ultimately it depends on the use case.

Thanks for the post! Great job.

joern Post author
2018-09-19 at 18:48

thanks, always open to further pointers...

leonado
2018-10-17 at 09:44

Thank your for your tutorial, it was very helpful!

I searched in the comments, and didn't find what I've been looking for:

1) suppose that, after plotting my dendrogram, the minimum high is 0.4, and the maximum is 1.
2) however, the plot will show the leaves starting at 0 (touching the x axis).
3) I'd like to plot without touching the x axis, but each leave starting from its minimal distance, as in the this figure https://goo.gl/images/LMiQUp

How could I do it? Is it possible with a simple argument at hierarchy.dendrogram(Z)? I tried several things and nothing...

RéMi
2018-10-23 at 11:03

Great, great article. I've learned a lot.

Would you have any comments regarding scipy.cluster.hierarchy.leaders function ? Can it be used to get the sample that is the most representative of each cluster (and therefore reduce drastically sample count) ?

jack
2018-11-08 at 17:23

Hi, thanks for sharing, could please tell me how to do this clustering on fmri 4d data?