Hierarchical Clustering

CS5154/6054

Yizong Cheng

11/10/2022

17 *Hierarchical clustering*

HIERARCHICAL CLUSTERING

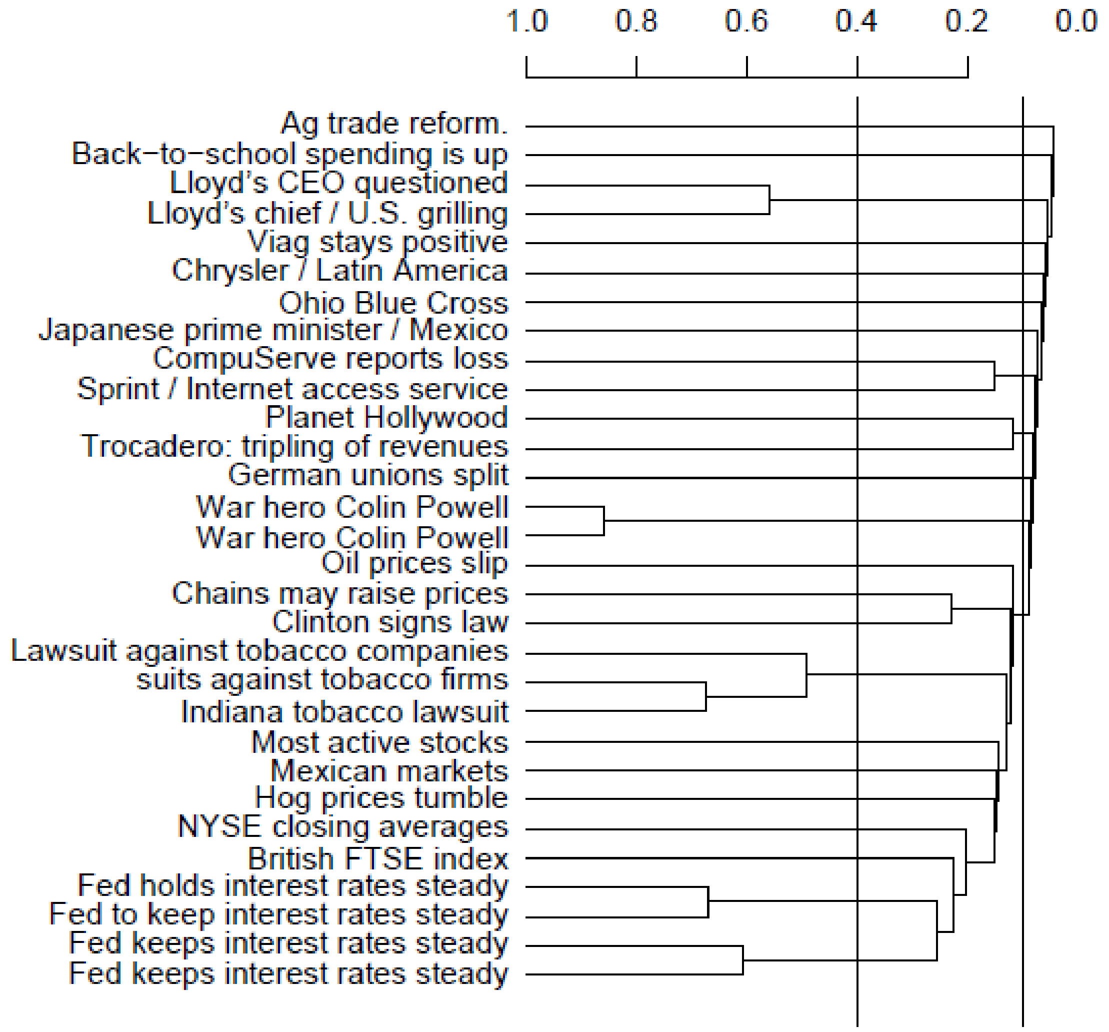
Flat clustering is efficient and conceptually simple, but as we saw in Chapter 16 it has a number of drawbacks. The algorithms introduced in Chapter 16 return a flat unstructured set of clusters, require a prespecified number of clusters as input and are nondeterministic. *Hierarchical clustering* (or *hierarchic clustering*) outputs a hierarchy, a structure that is more informative than the unstructured set of clusters returned by flat clustering.[1] Hierarchical clustering does not require us to prespecify the number of clusters and most hierarchical algorithms that have been used in IR are deterministic. These advantages of hierarchical clustering come at the cost of lower efficiency. The

17.1 Hierarchical agglomerative clustering

Hierarchical clustering algorithms are either top-down or bottom-up. Bottom-up algorithms treat each document as a singleton cluster at the outset and then successively merge (or *agglomerate*) pairs of clusters until all clusters have been merged into a single cluster that contains all documents. Bottom-up hierarchical clustering is therefore called *hierarchical agglomerative clustering* or *HAC*. Top-down clustering requires a method for splitting a cluster. It proceeds by splitting clusters recursively until individual documents are reached. See Section 17.6. HAC is more frequently used in IR than top-down clustering and is the main subject of this chapter.

HIERARCHICAL AGGLOMERATIVE CLUSTERING

HAC



▼ **Figure 17.1** A dendrogram of a single-link clustering of 30 documents from Reuters-RCV1. Two possible cuts of the dendrogram are shown: at 0.4 into 24 clusters and at 0.1 into 12 clusters.

DENDROGRAM

An HAC clustering is typically visualized as a *dendrogram* as shown in Figure 17.1. Each merge is represented by a horizontal line. The y-coordinate of the horizontal line is the similarity of the two clusters that were merged, where documents are viewed as singleton clusters. We call this similarity the *combination similarity* of the merged cluster. For example, the combination similarity of the cluster consisting of *Lloyd's CEO questioned* and *Lloyd's chief / U.S. grilling* in Figure 17.1 is ≈ 0.56. We define the combination similarity of a singleton cluster as its document's self-similarity (which is 1.0 for cosine similarity).

COMBINATION SIMILARITY

By moving up from the bottom layer to the top node, a dendrogram allows us to reconstruct the history of merges that resulted in the depicted clustering. For example, we see that the two documents entitled *War hero Colin Powell* were merged first in Figure 17.1 and that the last merge added *Ag trade reform* to a cluster consisting of the other 29 documents.

MONOTONICITY

A fundamental assumption in HAC is that the merge operation is *monotonic*. Monotonic means that if $s_1, s_2, \ldots, s_{K-1}$ are the combination similarities of the successive merges of an HAC, then $s_1 \geq s_2 \geq \ldots \geq s_{K-1}$ holds. A non-

```
SimpleHAC($d_1, \ldots, d_N$)
 1  for $n \leftarrow 1$ to $N$
 2  do for $i \leftarrow 1$ to $N$
 3       do $C[n][i] \leftarrow \textsc{Sim}(d_n, d_i)$
 4       $I[n] \leftarrow 1$  (keeps track of active clusters)
 5  $A \leftarrow []$  (assembles clustering as a sequence of merges)
 6  for $k \leftarrow 1$ to $N - 1$
 7  do $\langle i, m \rangle \leftarrow \arg\max_{\{\langle i,m \rangle : i \neq m \wedge I[i]=1 \wedge I[m]=1\}} C[i][m]$
 8       $A.\textsc{Append}(\langle i, m \rangle)$  (store merge)
 9       for $j \leftarrow 1$ to $N$
10       do $C[i][j] \leftarrow \textsc{Sim}(i, m, j)$
11            $C[j][i] \leftarrow \textsc{Sim}(i, m, j)$
12       $I[m] \leftarrow 0$  (deactivate cluster)
13  return $A$
```

▶ **Figure 17.2** A simple, but inefficient HAC algorithm.

A simple, naive HAC algorithm is shown in Figure 17.2. We first compute the $N \times N$ similarity matrix C. The algorithm then executes $N-1$ steps of merging the currently most similar clusters. In each iteration, the two most similar clusters are merged and the rows and columns of the merged cluster i in C are updated.[2] The clustering is stored as a list of merges in A. I indicates which clusters are still available to be merged. The function $\text{SIM}(i, m, j)$ computes the similarity of cluster j with the merge of clusters i and m. For some HAC algorithms, $\text{SIM}(i, m, j)$ is simply a function of $C[j][i]$ and $C[j][m]$, for example, the maximum of these two values for single-link.

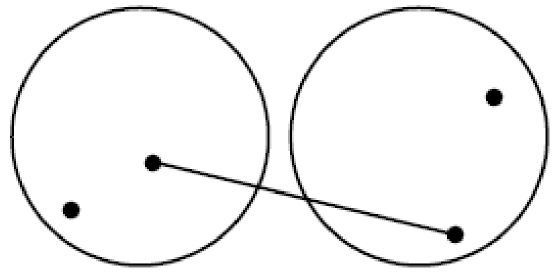
17.2 Single-link and complete-link clustering

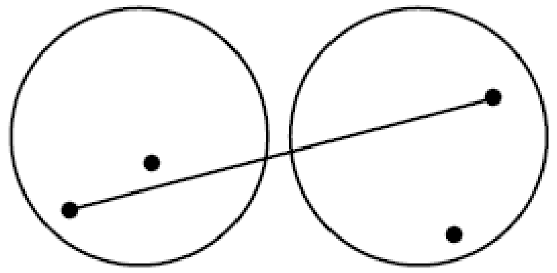
SINGLE-LINK CLUSTERING

In *single-link clustering* or *single-linkage clustering*, the similarity of two clusters is the similarity of their *most similar* members (see Figure 17.3, (a))[3]. This single-link merge criterion is *local*. We pay attention solely to the area where the two clusters come closest to each other. Other, more distant parts of the cluster and the clusters' overall structure are not taken into account.

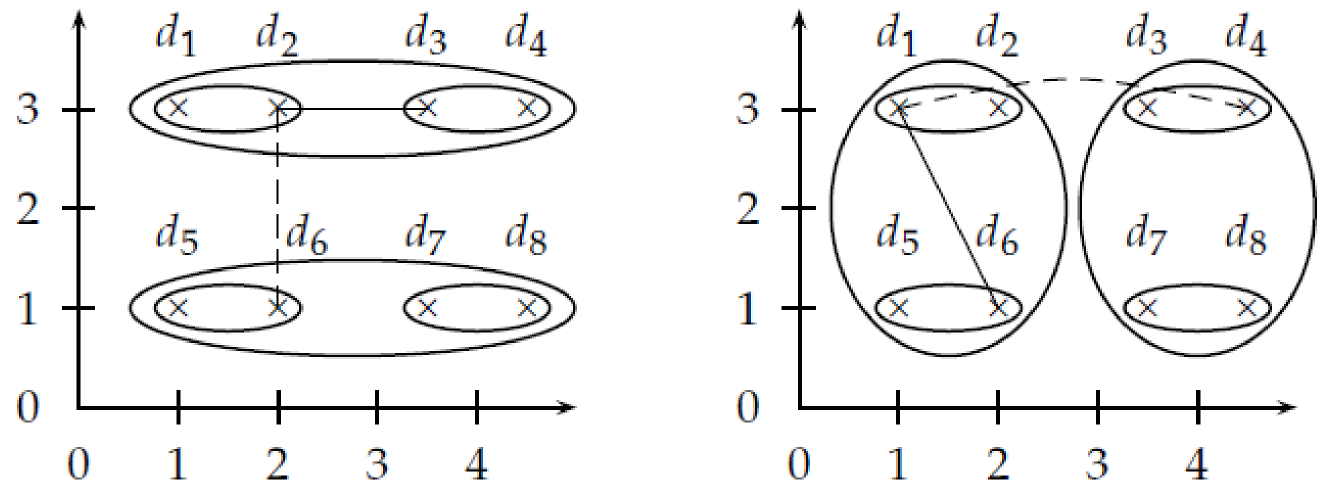
COMPLETE-LINK CLUSTERING

In *complete-link clustering* or *complete-linkage clustering*, the similarity of two clusters is the similarity of their *most dissimilar* members (see Figure 17.3, (b)). This is equivalent to choosing the cluster pair whose merge has the smallest diameter. This complete-link merge criterion is non-local; the entire structure of the clustering can influence merge decisions. This results in a preference for compact clusters with small diameters over long, straggly clusters, but also causes sensitivity to outliers. A single document far from the center can increase diameters of candidate merge clusters dramatically and completely change the final clustering.

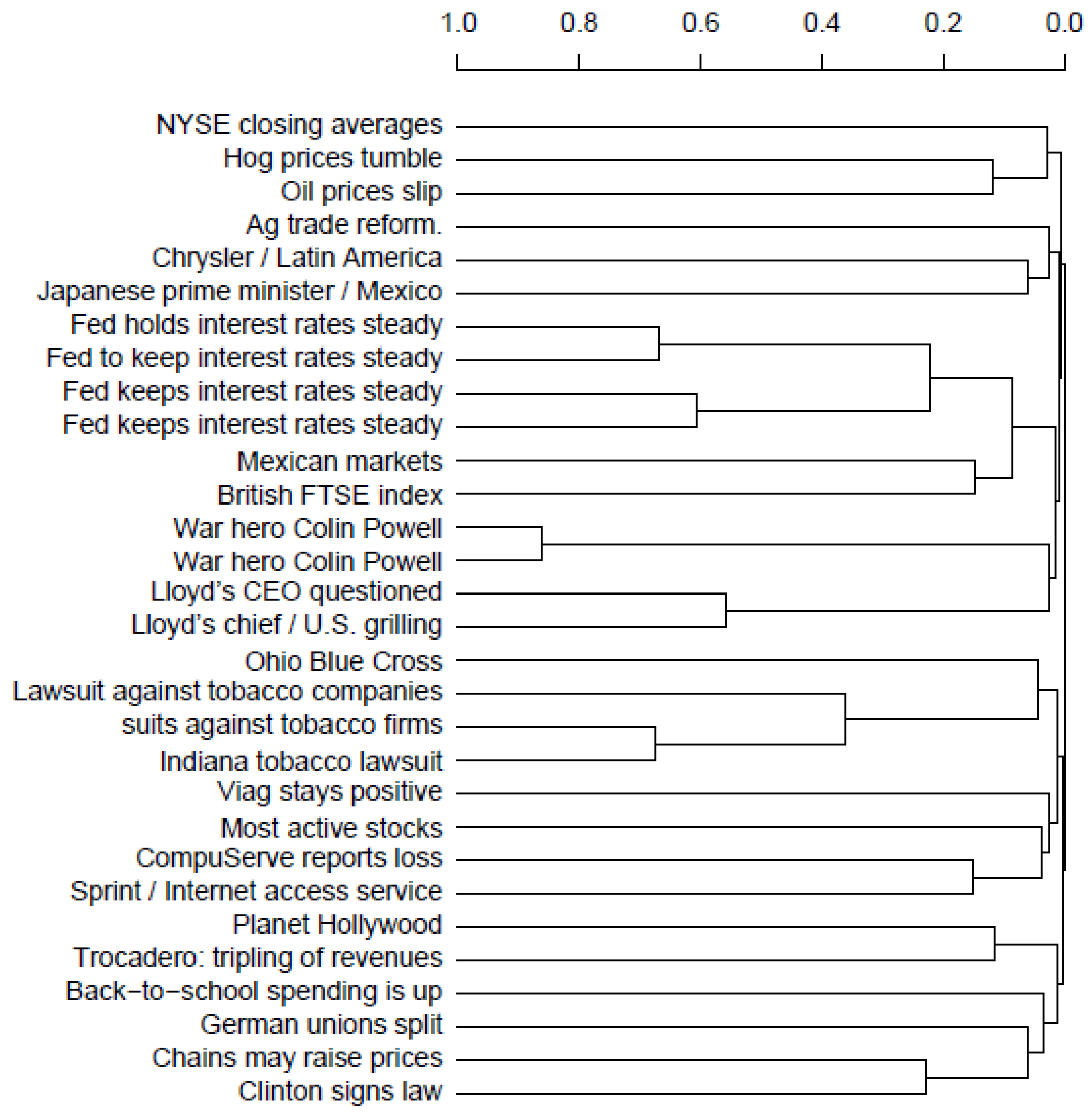

(a) single-link: maximum similarity


(b) complete-link: minimum similarity

clustering algorithm	$\text{SIM}(i, k_1, k_2)$
single-link	$\max(\text{SIM}(i, k_1), \text{SIM}(i, k_2))$
complete-link	$\min(\text{SIM}(i, k_1), \text{SIM}(i, k_2))$



▶ **Figure 17.4** A single-link (left) and complete-link (right) clustering of eight documents. The ellipses correspond to successive clustering stages. Left: The single-link similarity of the two upper two-point clusters is the similarity of d_2 and d_3 (solid line), which is greater than the single-link similarity of the two left two-point clusters (dashed line). Right: The complete-link similarity of the two upper two-point clusters is the similarity of d_1 and d_4 (dashed line), which is smaller than the complete-link similarity of the two left two-point clusters (solid line).



▼ **Figure 17.5** A dendrogram of a complete-link clustering. The same 30 documents were clustered with single-link clustering in Figure 17.1.

```
EFFICIENTHAC($\vec{d}_1, \ldots, \vec{d}_N$)
 1  for $n \leftarrow 1$ to $N$
 2  do for $i \leftarrow 1$ to $N$
 3       do $C[n][i].\text{sim} \leftarrow \vec{d}_n \cdot \vec{d}_i$
 4          $C[n][i].\text{index} \leftarrow i$
 5       $I[n] \leftarrow 1$
 6       $P[n] \leftarrow$ priority queue for $C[n]$ sorted on sim
 7       $P[n]$.DELETE($C[n][n]$) (don't want self-similarities)
 8  $A \leftarrow []$
 9  for $k \leftarrow 1$ to $N - 1$
10  do $k_1 \leftarrow \arg\max_{\{k:I[k]=1\}} P[k].\text{MAX}().\text{sim}$
11       $k_2 \leftarrow P[k_1].\text{MAX}().\text{index}$
12       $A$.APPEND($\langle k_1, k_2 \rangle$)
13       $I[k_2] \leftarrow 0$
14       $P[k_1] \leftarrow []$
15       for each $i$ with $I[i] = 1 \wedge i \neq k_1$
16       do $P[i]$.DELETE($C[i][k_1]$)
17          $P[i]$.DELETE($C[i][k_2]$)
18          $C[i][k_1].\text{sim} \leftarrow \text{SIM}(i, k_1, k_2)$
19          $P[i]$.INSERT($C[i][k_1]$)
20          $C[k_1][i].\text{sim} \leftarrow \text{SIM}(i, k_1, k_2)$
21          $P[k_1]$.INSERT($C[k_1][i]$)
22  return $A$
```

```
SINGLELINKCLUSTERING($d_1, \ldots, d_N$)
for $n \leftarrow 1$ to $N$
do for $i \leftarrow 1$ to $N$
    do $C[n][i].\text{sim} \leftarrow \text{SIM}(d_n, d_i)$
        $C[n][i].\text{index} \leftarrow i$
    $I[n] \leftarrow n$
    $NBM[n] \leftarrow \arg\max_{X \in \{C[n][i]: n \neq i\}} X.\text{sim}$
$A \leftarrow []$
for $n \leftarrow 1$ to $N - 1$
do $i_1 \leftarrow \arg\max_{\{i: I[i]=i\}} NBM[i].\text{sim}$
    $i_2 \leftarrow I[NBM[i_1].\text{index}]$
    $A.\text{APPEND}(\langle i_1, i_2 \rangle)$
    for $i \leftarrow 1$ to $N$
    do if $I[i] = i \wedge i \neq i_1 \wedge i \neq i_2$
          then $C[i_1][i].\text{sim} \leftarrow C[i][i_1].\text{sim} \leftarrow \max(C[i_1][i].\text{sim}, C[i_2][i].\text{sim})$
        if $I[i] = i_2$
          then $I[i] \leftarrow i_1$
    $NBM[i_1] \leftarrow \arg\max_{X \in \{C[i_1][i]: I[i]=i \wedge i \neq i_1\}} X.\text{sim}$
return $A$
```

▶ **Figure 17.9** Single-link clustering algorithm using an NBM array. After merging two clusters i_1 and i_2, the first one (i_1) represents the merged cluster. If $I[i] = i$, then i is the representative of its current cluster. If $I[i] \neq i$, then i has been merged into the cluster represented by $I[i]$ and will therefore be ignored when updating $NBM[i_1]$.

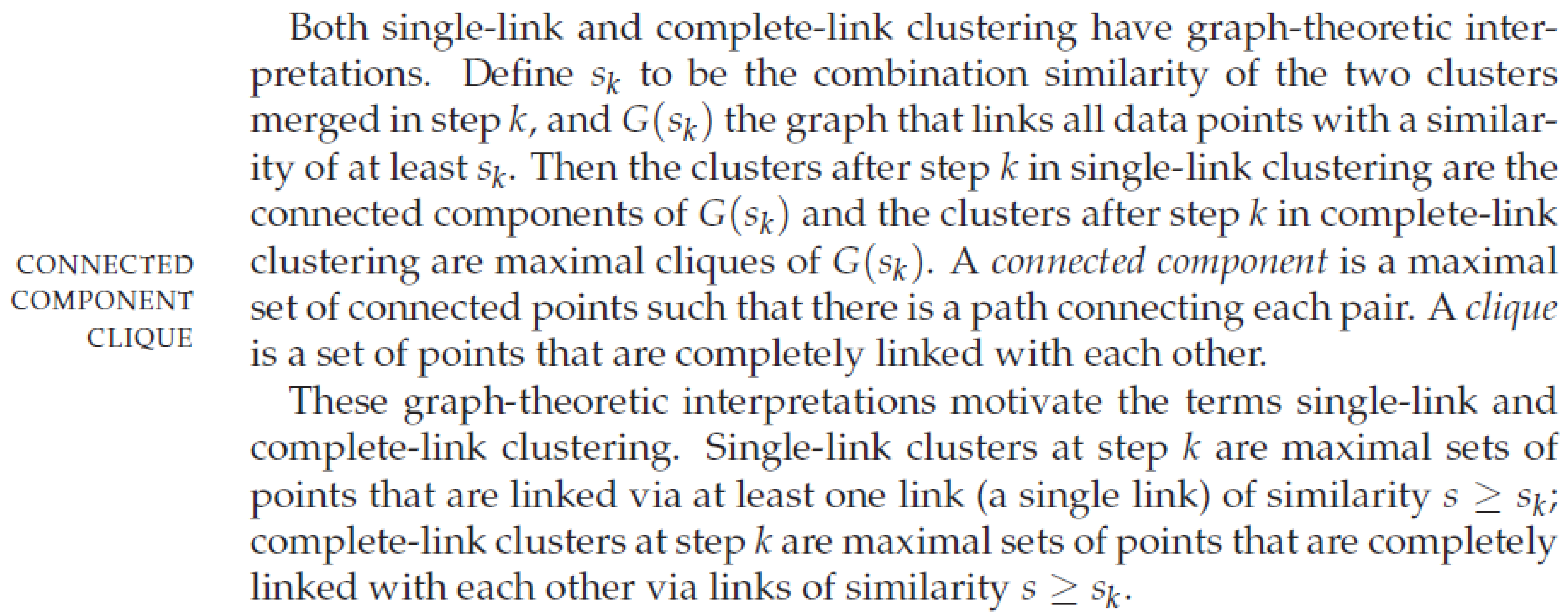
KRUSKAL'S ALGORITHM

large variety of hierarchical clustering algorithms). The single-link algorithm in Figure 17.9 is similar to *Kruskal's algorithm* for constructing a minimum spanning tree. A graph-theoretical proof of the correctness of Kruskal's algorithm (which is analogous to the proof in Section 17.5) is provided by Cormen et al. (1990, Theorem 23.1). See Exercise 17.5 for the connection between minimum spanning trees and single-link clusterings.

Exercise 17.5

A single-link clustering can also be computed from the *minimum spanning tree* of a graph. The minimum spanning tree connects the vertices of a graph at the smallest possible cost, where cost is defined as the sum over all edges of the graph. In our case the cost of an edge is the distance between two documents. Show that if $\Delta_{k-1} > \Delta_k > \ldots > \Delta_1$ are the costs of the edges of a minimum spanning tree, then these edges correspond to the $k-1$ merges in constructing a single-link clustering.

Graph-Theoretical Interpretations



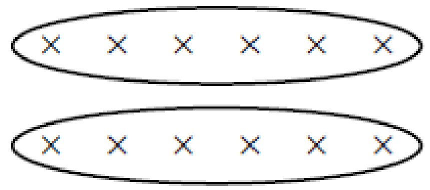
Both single-link and complete-link clustering have graph-theoretic interpretations. Define s_k to be the combination similarity of the two clusters merged in step k, and $G(s_k)$ the graph that links all data points with a similarity of at least s_k. Then the clusters after step k in single-link clustering are the connected components of $G(s_k)$ and the clusters after step k in complete-link clustering are maximal cliques of $G(s_k)$. A *connected component* is a maximal set of connected points such that there is a path connecting each pair. A *clique* is a set of points that are completely linked with each other.

CONNECTED COMPONENT
CLIQUE

These graph-theoretic interpretations motivate the terms single-link and complete-link clustering. Single-link clusters at step k are maximal sets of points that are linked via at least one link (a single link) of similarity $s \geq s_k$; complete-link clusters at step k are maximal sets of points that are completely linked with each other via links of similarity $s \geq s_k$.

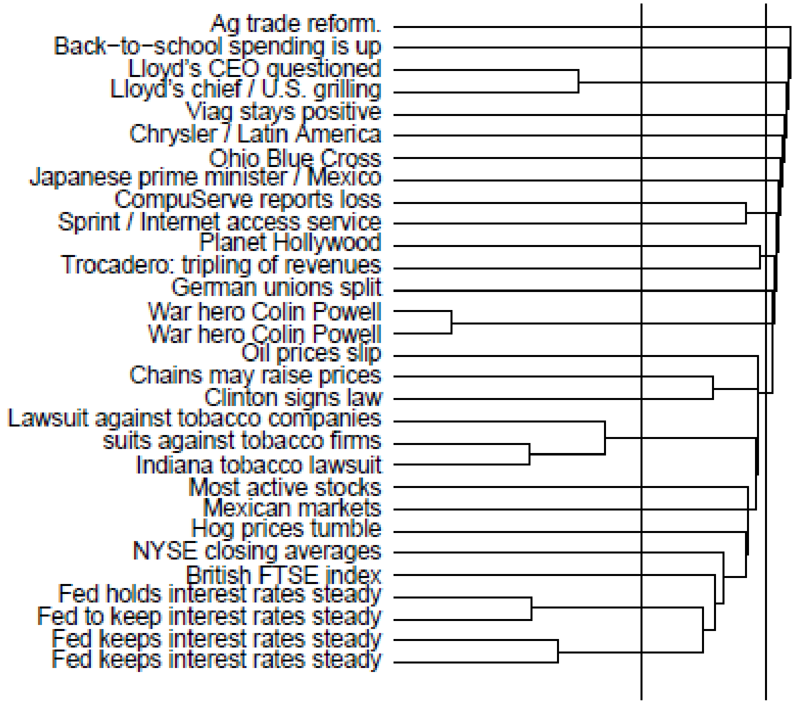
Single-link and complete-link clustering reduce the assessment of cluster quality to a single similarity between a pair of documents: the two most similar documents in single-link clustering and the two most dissimilar documents in complete-link clustering. A measurement based on one pair cannot fully reflect the distribution of documents in a cluster. It is therefore not surprising that both algorithms often produce undesirable clusters. Single-link clustering can produce straggling clusters as shown in Figure 17.6. Since the merge criterion is strictly local, a chain of points can be extended for long distances without regard to the overall shape of the emerging cluster. This effect is called *chaining*.

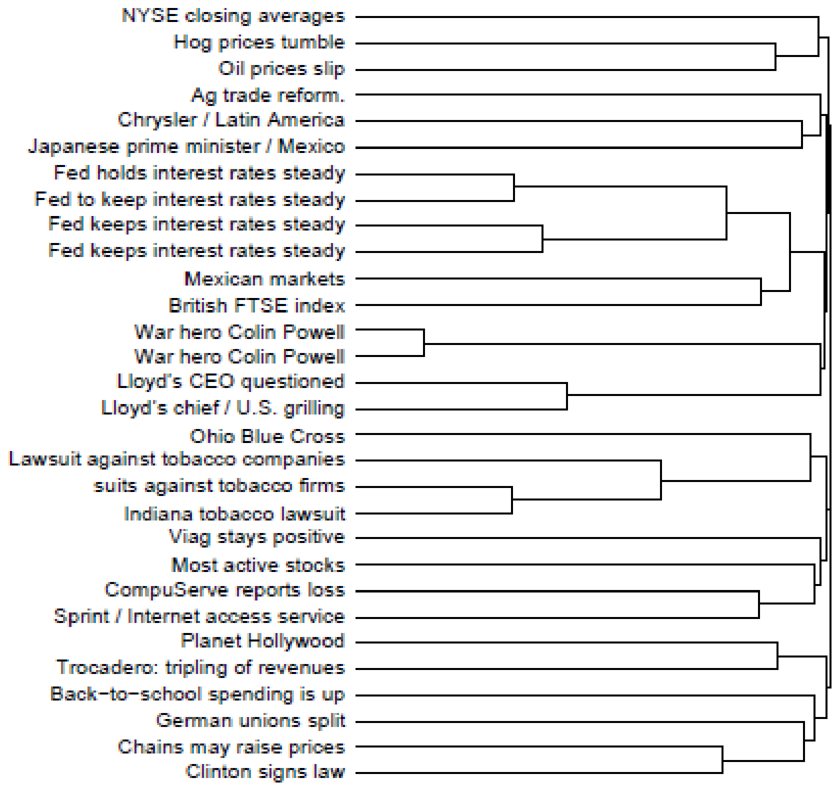
CHAINING



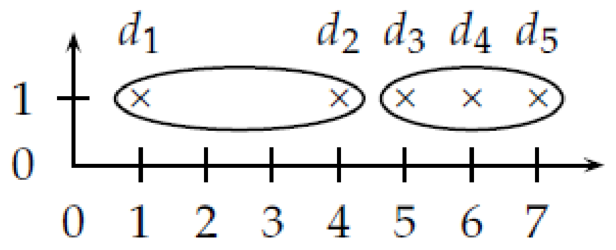
▶ **Figure 17.6** Chaining in single-link clustering. The local criterion in single-link clustering can cause undesirable elongated clusters.

The chaining effect is also apparent in Figure 17.1. The last eleven merges of the single-link clustering (those above the 0.1 line) add on single documents or pairs of documents, corresponding to a chain. The complete-link clustering in Figure 17.5 avoids this problem. Documents are split into two groups of roughly equal size when we cut the dendrogram at the last merge. In general, this is a more useful organization of the data than a clustering with chains.

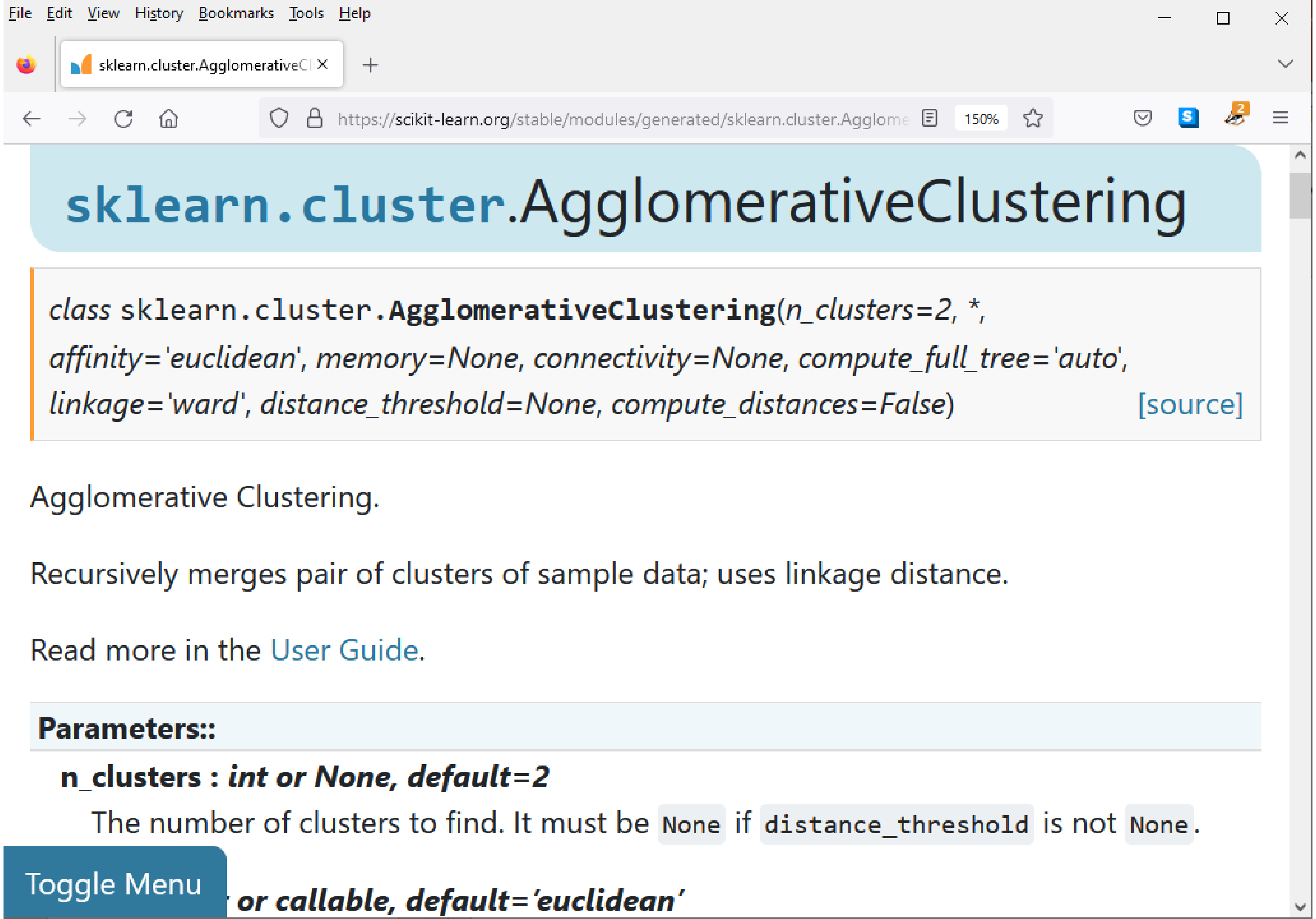




However, complete-link clustering suffers from a different problem. It pays too much attention to outliers, points that do not fit well into the global structure of the cluster. In the example in Figure 17.7 the four documents d_2, d_3, d_4, d_5 are split because of the outlier d_1 at the left edge (Exercise 17.1). Complete-link clustering does not find the most intuitive cluster structure in



▶ **Figure 17.7** Outliers in complete-link clustering. The five documents have the x-coordinates $1 + 2\epsilon, 4, 5 + 2\epsilon, 6$ and $7 - \epsilon$. Complete-link clustering creates the two clusters shown as ellipses. The most intuitive two-cluster clustering is $\{\{d_1\}, \{d_2, d_3, d_4, d_5\}\}$, but in complete-link clustering, the outlier d_1 splits $\{d_2, d_3, d_4, d_5\}$ as shown.


File Edit View History Bookmarks Tools Help
sklearn.cluster.AgglomerativeCl
https://scikit-learn.org/stable/modules/generated/sklearn.cluster.Agglome
150%
Toggle Menu

sklearn.cluster.AgglomerativeClustering

class `sklearn.cluster.`**`AgglomerativeClustering`**(*n_clusters=2, *, affinity='euclidean', memory=None, connectivity=None, compute_full_tree='auto', linkage='ward', distance_threshold=None, compute_distances=False*) [source]

Agglomerative Clustering.

Recursively merges pair of clusters of sample data; uses linkage distance.

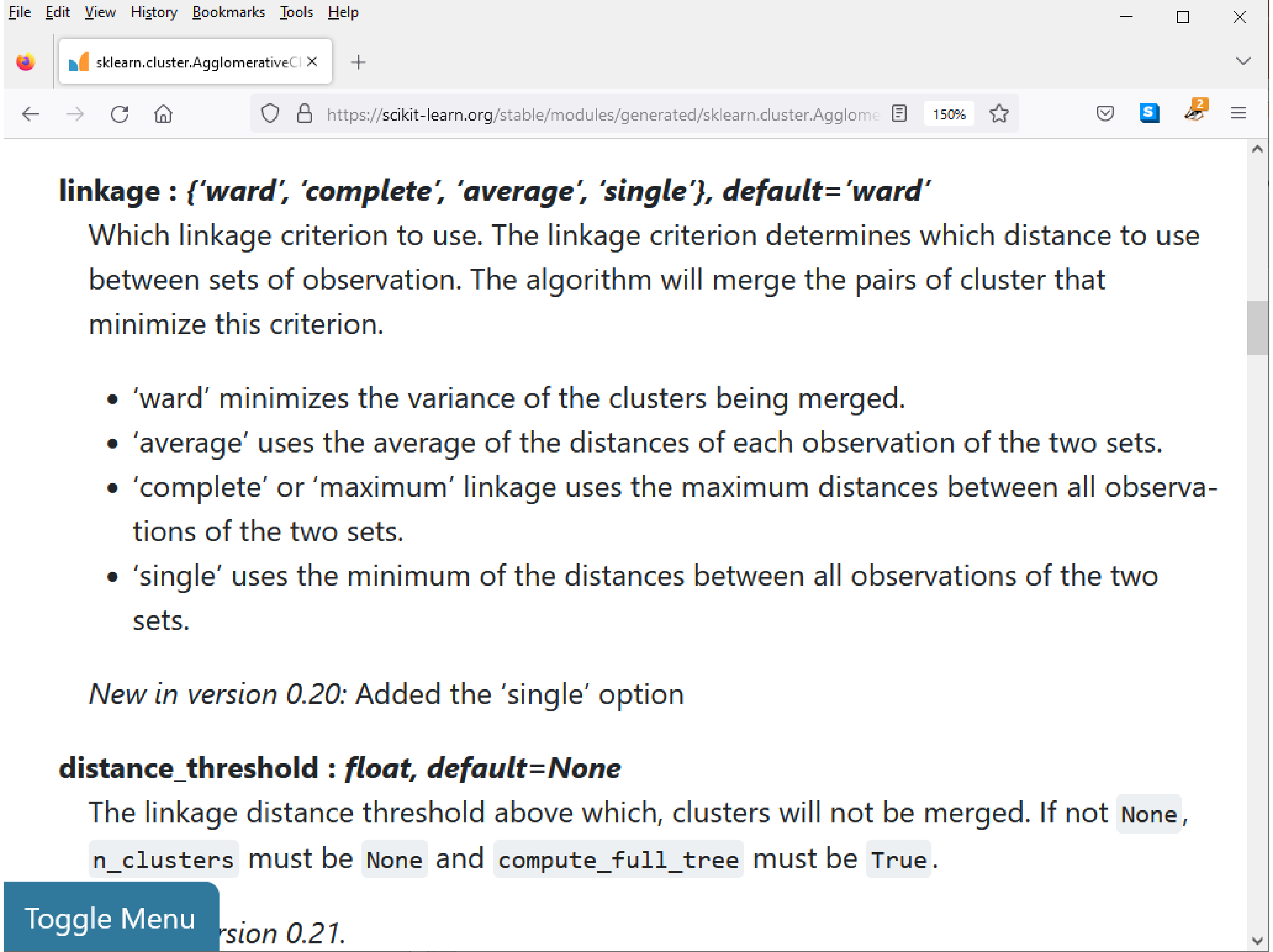
Read more in the User Guide.

Parameters::

n_clusters : *int or None, default=2*

The number of clusters to find. It must be `None` if `distance_threshold` is not `None`.

or callable, default='euclidean'


File Edit View History Bookmarks Tools Help
sklearn.cluster.AgglomerativeCl
https://scikit-learn.org/stable/modules/generated/sklearn.cluster.Agglome
150%
Toggle Menu

linkage : *{'ward', 'complete', 'average', 'single'}, default='ward'*

Which linkage criterion to use. The linkage criterion determines which distance to use between sets of observation. The algorithm will merge the pairs of cluster that minimize this criterion.

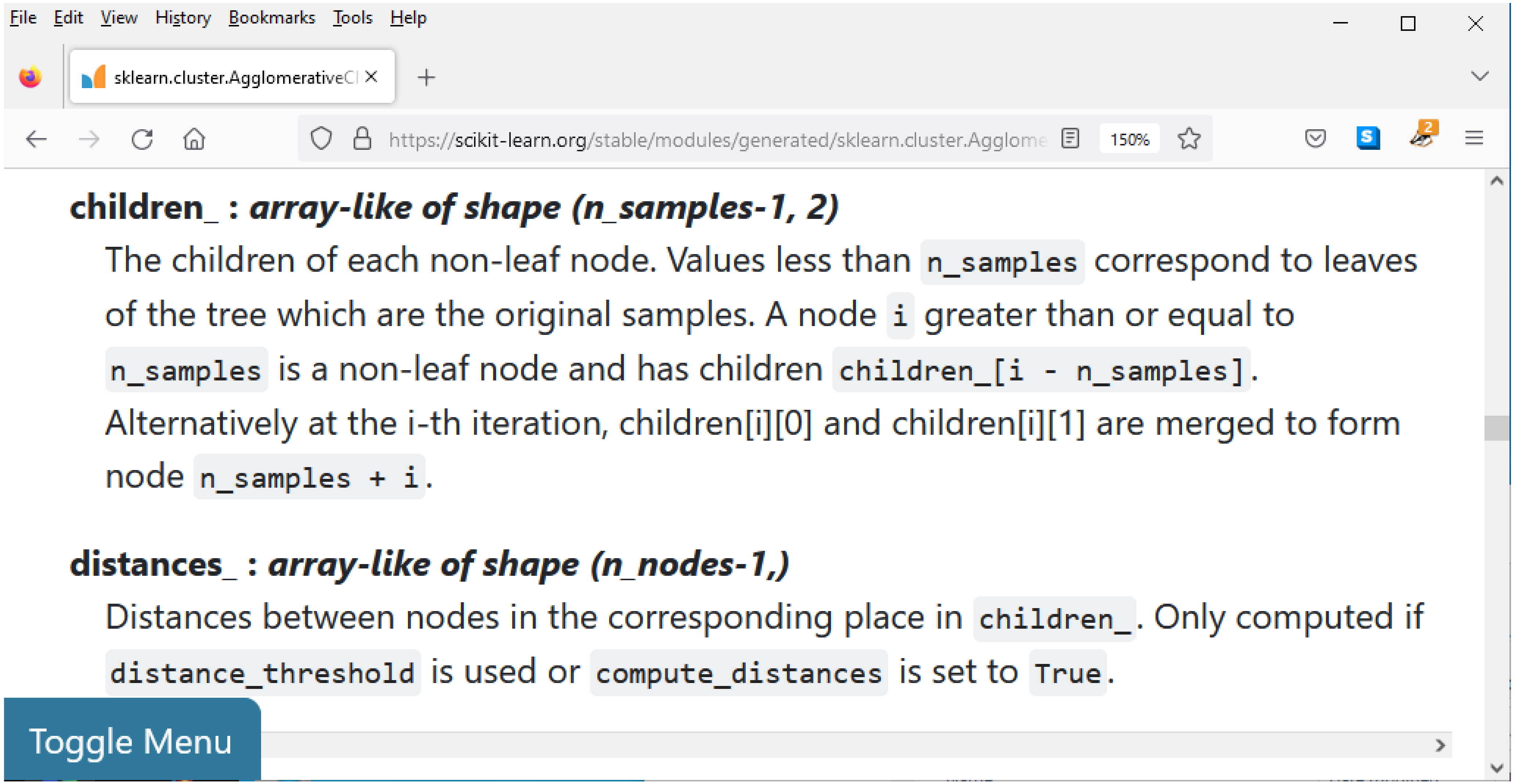
- 'ward' minimizes the variance of the clusters being merged.
- 'average' uses the average of the distances of each observation of the two sets.
- 'complete' or 'maximum' linkage uses the maximum distances between all observations of the two sets.
- 'single' uses the minimum of the distances between all observations of the two sets.

New in version 0.20: Added the 'single' option

distance_threshold : *float, default=None*

The linkage distance threshold above which, clusters will not be merged. If not `None`, `n_clusters` must be `None` and `compute_full_tree` must be `True`.

rsion 0.21.


File Edit View History Bookmarks Tools Help
sklearn.cluster.AgglomerativeCl
https://scikit-learn.org/stable/modules/generated/sklearn.cluster.Agglome
150%

children_ : *array-like of shape (n_samples-1, 2)*

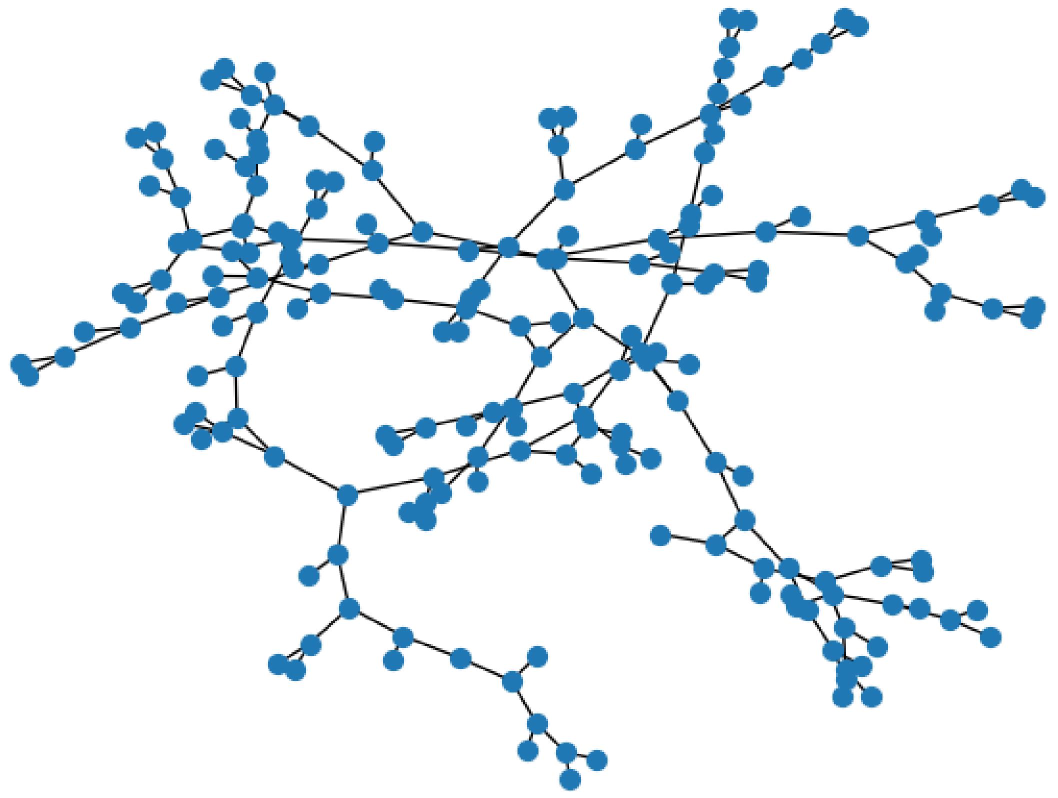
The children of each non-leaf node. Values less than `n_samples` correspond to leaves of the tree which are the original samples. A node `i` greater than or equal to `n_samples` is a non-leaf node and has children `children_[i - n_samples]`. Alternatively at the i-th iteration, children[i][0] and children[i][1] are merged to form node `n_samples + i`.

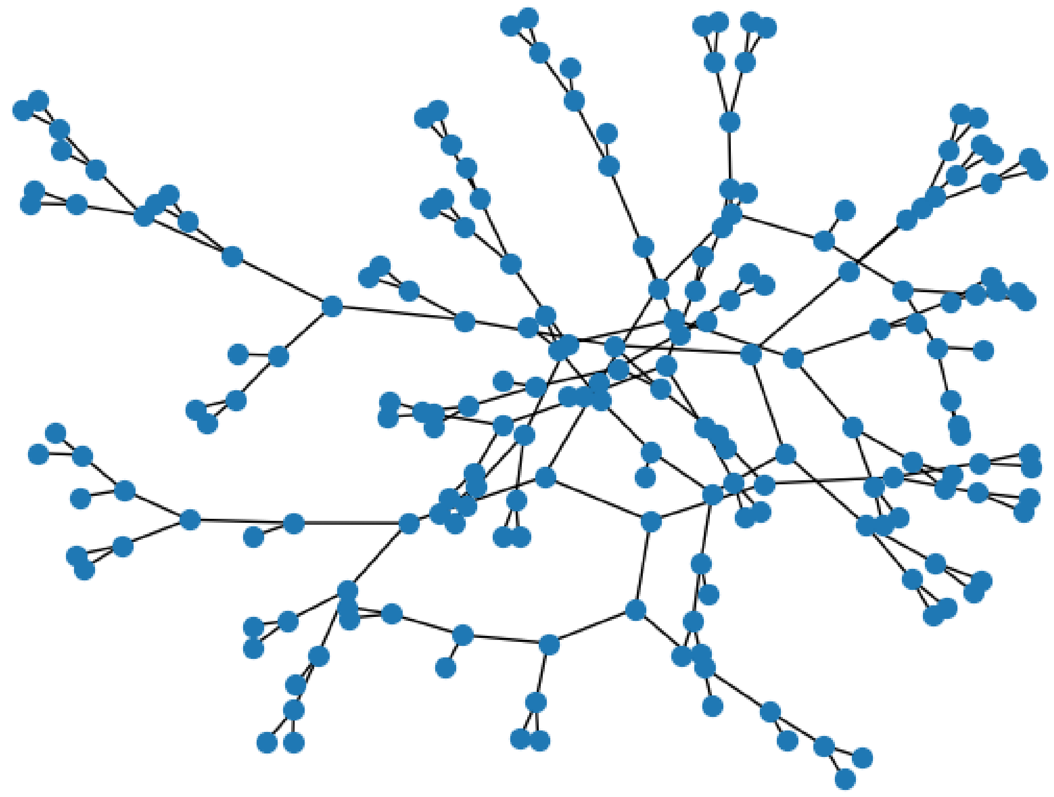
distances_ : *array-like of shape (n_nodes-1,)*

Distances between nodes in the corresponding place in `children_`. Only computed if `distance_threshold` is used or `compute_distances` is set to `True`.

```
from matplotlib import pyplot as plt
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.cluster import AgglomerativeClustering

f = open("bible.txt", "r")
docs = f.readlines()
f.close()
N = 100
firstk = docs[0:N]

cv = TfidfVectorizer(max_df=0.4, min_df=2)
X = cv.fit_transform(firstk).toarray()

model = AgglomerativeClustering(distance_threshold=0, n_clusters=None, linkage='single')
G = nx.Graph()
model = model.fit(X)
for i in range(N - 1):
   G.add_edge(N + i, model.children_[i][0])
   G.add_edge(N + i, model.children_[i][1])
nx.draw(G, node_size=50)
plt.show()
```

Single (left) and Complete (right) Linkage



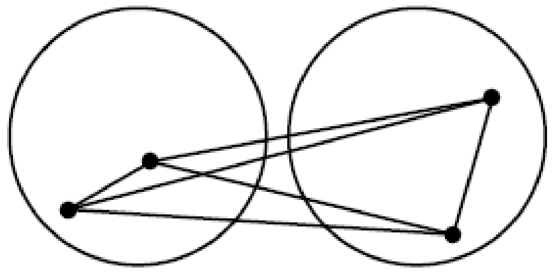


17.3 Group-average agglomerative clustering

GROUP-AVERAGE AGGLOMERATIVE CLUSTERING

Group-average agglomerative clustering or *GAAC* (see Figure 17.3, (d)) evaluates cluster quality based on *all* similarities between documents, thus avoiding the pitfalls of the single-link and complete-link criteria, which equate cluster similarity with the similarity of a single pair of documents. GAAC is also called *group-average clustering* and *average-link clustering*. GAAC computes the average similarity SIM-GA of all pairs of documents, including pairs from the same cluster. But self-similarities are not included in the average:

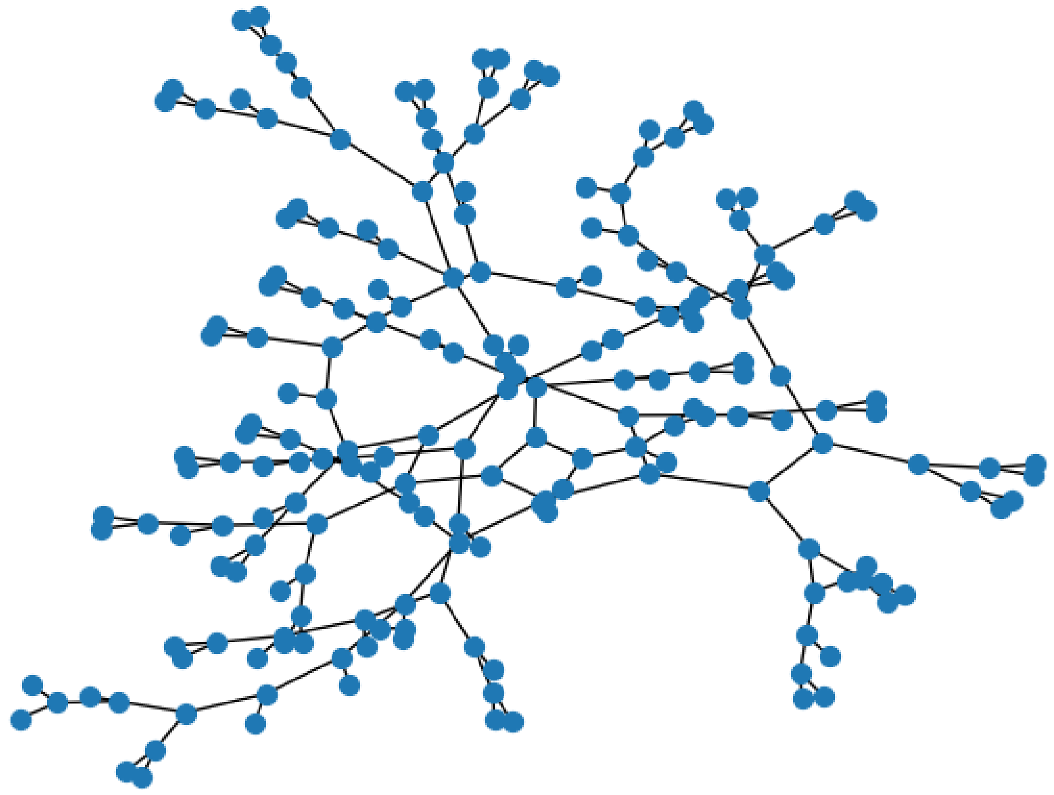
$$\text{SIM-GA}(\omega_i, \omega_j) = \frac{1}{(N_i + N_j)(N_i + N_j - 1)} \sum_{d_m \in \omega_i \cup \omega_j} \sum_{d_n \in \omega_i \cup \omega_j, d_n \neq d_m} \vec{d}_m \cdot \vec{d}_n \tag{17.1}$$

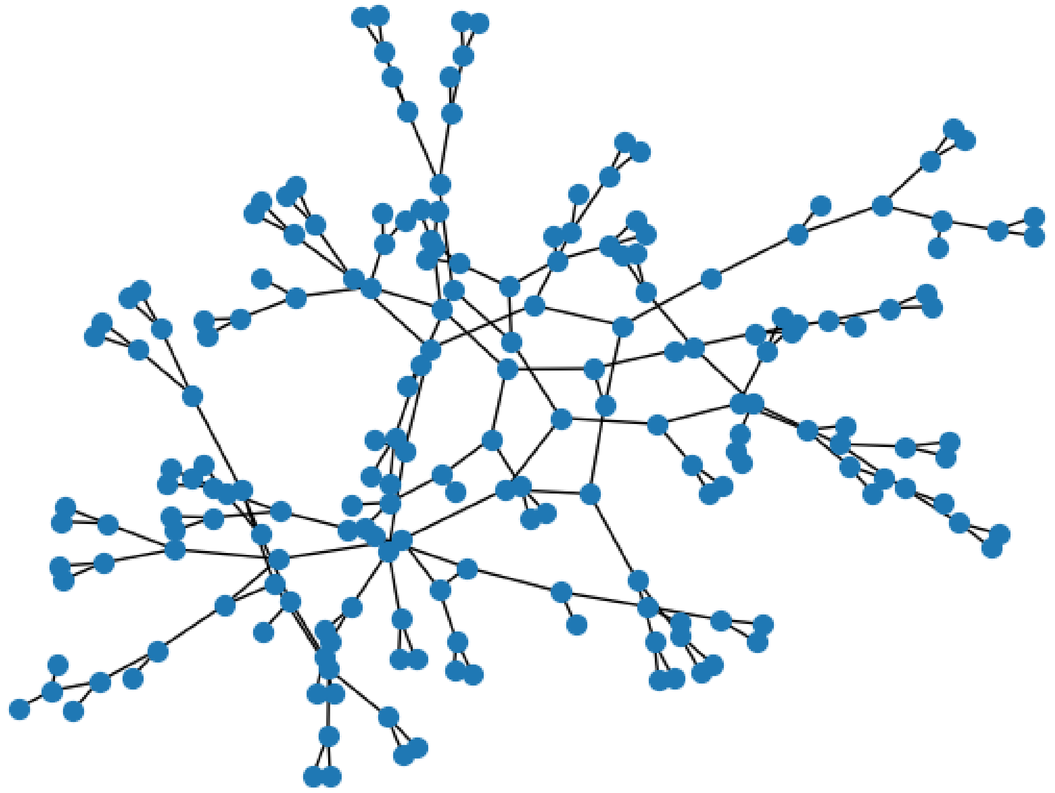


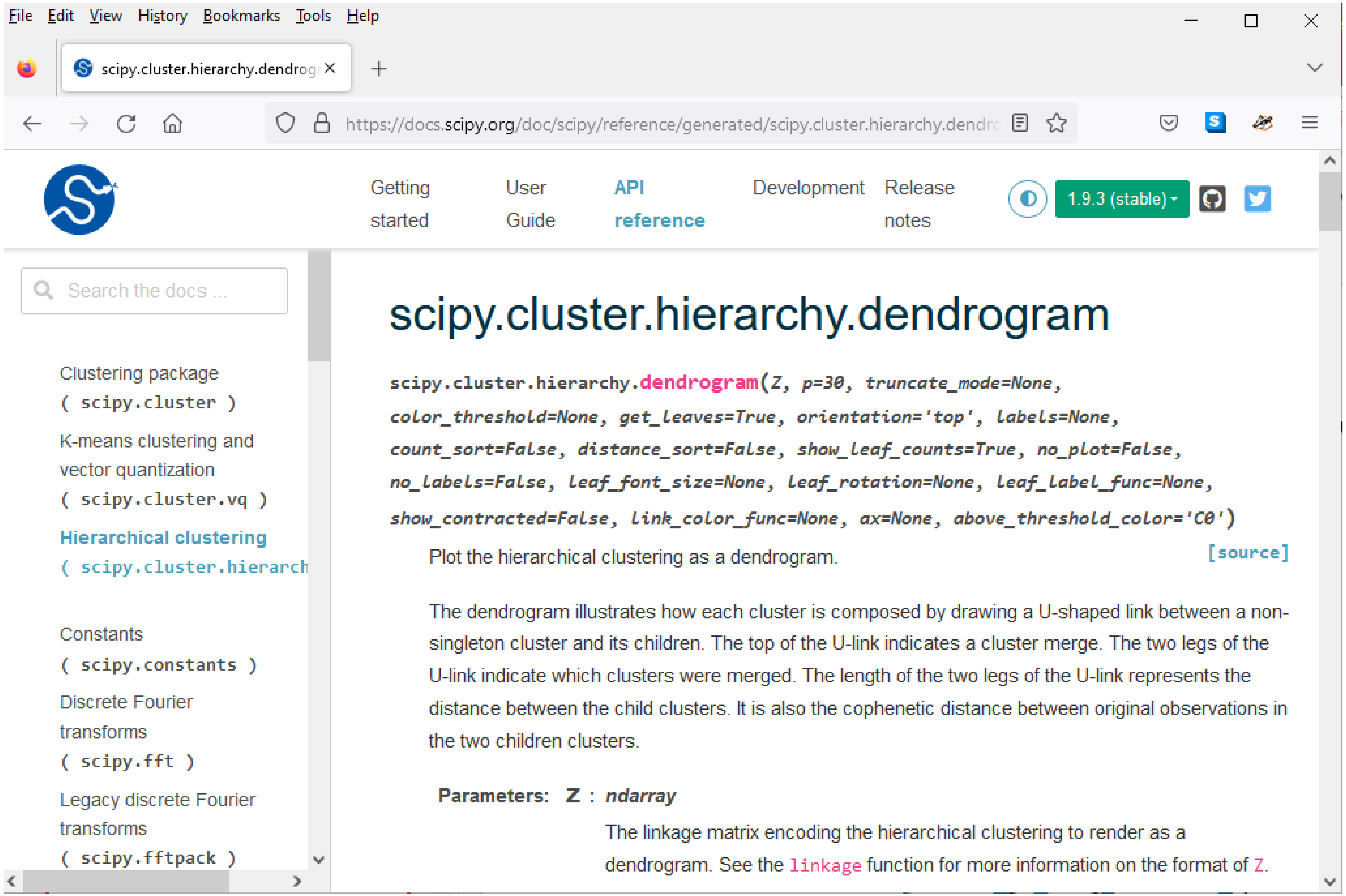
WARD'S METHOD

An important HAC technique not discussed here is *Ward's method* (Ward Jr. 1963, El-Hamdouchi and Willett 1986), also called *minimum variance clustering*. In each step, it selects the merge with the smallest RSS (Chapter 16, page 360). The merge criterion in Ward's method (a function of all individual distances from the centroid) is closely related to the merge criterion in GAAC (a function of all individual similarities to the centroid).

Average (left) and Ward (right) Linkage






File Edit View History Bookmarks Tools Help
scipy.cluster.hierarchy.dendrog
https://docs.scipy.org/doc/scipy/reference/generated/scipy.cluster.hierarchy.dendro
Getting started
User Guide
API reference
Development
Release notes
1.9.3 (stable)
Search the docs ...
Clustering package (scipy.cluster)
K-means clustering and vector quantization (scipy.cluster.vq)
Hierarchical clustering (scipy.cluster.hierarch
Constants (scipy.constants)
Discrete Fourier transforms (scipy.fft)
Legacy discrete Fourier transforms (scipy.fftpack)

scipy.cluster.hierarchy.dendrogram

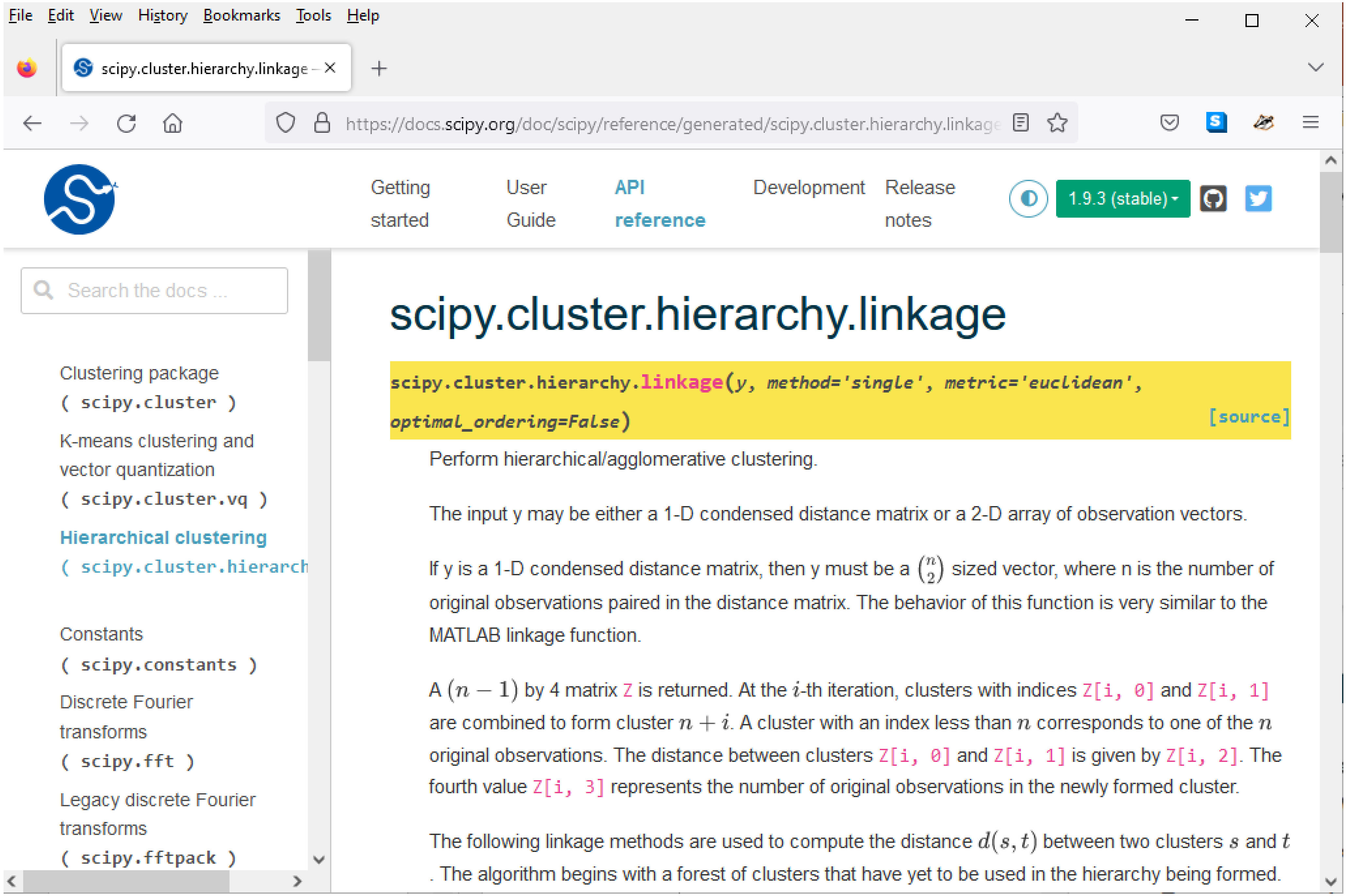
```
scipy.cluster.hierarchy.dendrogram(Z, p=30, truncate_mode=None,
color_threshold=None, get_leaves=True, orientation='top', labels=None,
count_sort=False, distance_sort=False, show_leaf_counts=True, no_plot=False,
no_labels=False, leaf_font_size=None, leaf_rotation=None, leaf_label_func=None,
show_contracted=False, link_color_func=None, ax=None, above_threshold_color='C0')
```

Plot the hierarchical clustering as a dendrogram. [source]

The dendrogram illustrates how each cluster is composed by drawing a U-shaped link between a non-singleton cluster and its children. The top of the U-link indicates a cluster merge. The two legs of the U-link indicate which clusters were merged. The length of the two legs of the U-link represents the distance between the child clusters. It is also the cophenetic distance between original observations in the two children clusters.

Parameters: **Z** : *ndarray*

The linkage matrix encoding the hierarchical clustering to render as a dendrogram. See the `linkage` function for more information on the format of Z.


File Edit View History Bookmarks Tools Help
scipy.cluster.hierarchy.linkage
https://docs.scipy.org/doc/scipy/reference/generated/scipy.cluster.hierarchy.linkage
Getting started
User Guide
API reference
Development
Release notes
1.9.3 (stable)
Search the docs ...

Clustering package (`scipy.cluster`)

K-means clustering and vector quantization (`scipy.cluster.vq`)

Hierarchical clustering (`scipy.cluster.hierarch`

Constants (`scipy.constants`)

Discrete Fourier transforms (`scipy.fft`)

Legacy discrete Fourier transforms (`scipy.fftpack`)

scipy.cluster.hierarchy.linkage

`scipy.cluster.hierarchy.linkage(y, method='single', metric='euclidean', optimal_ordering=False)` [source]

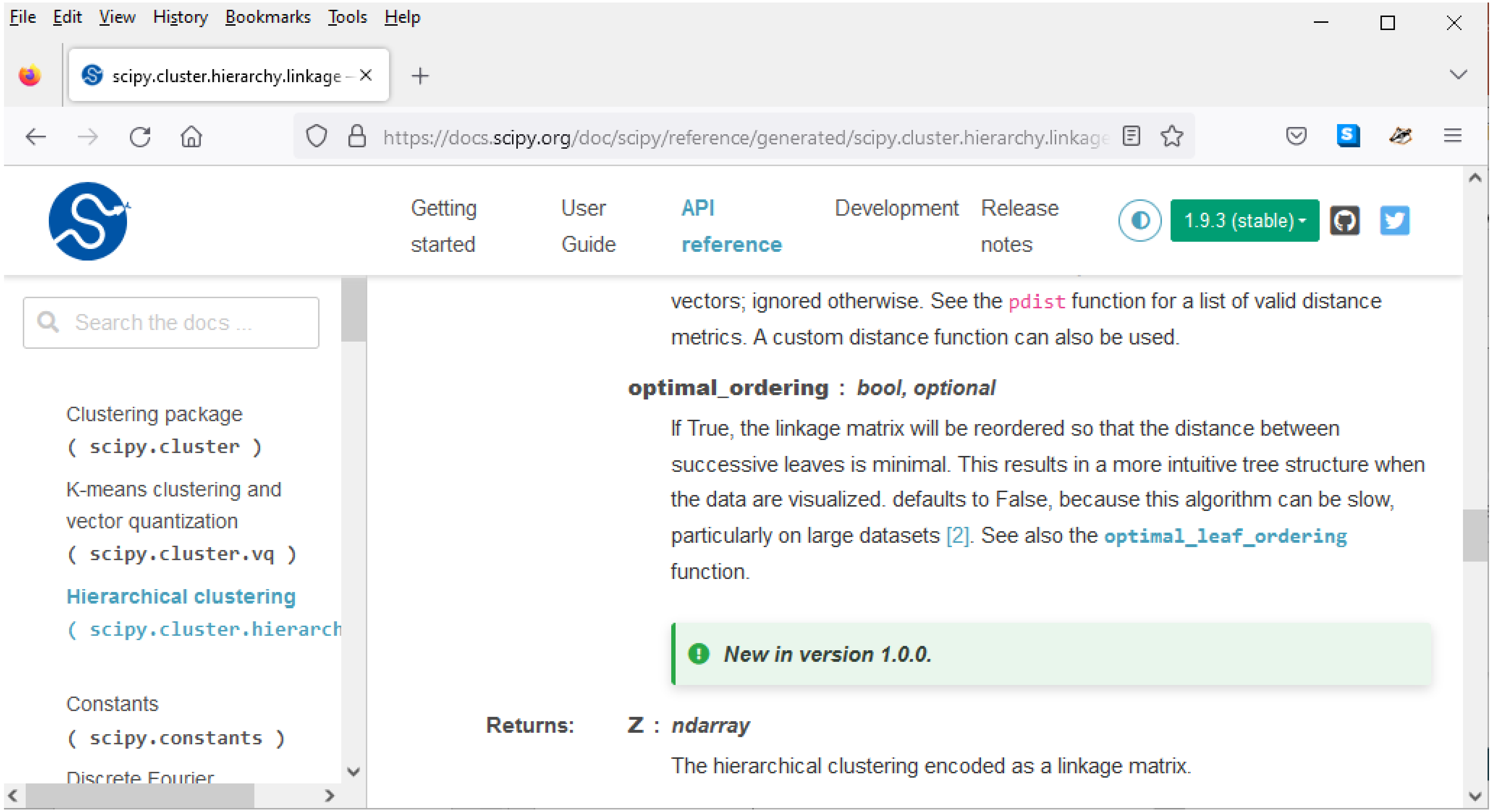
Perform hierarchical/agglomerative clustering.

The input y may be either a 1-D condensed distance matrix or a 2-D array of observation vectors.

If y is a 1-D condensed distance matrix, then y must be a $\binom{n}{2}$ sized vector, where n is the number of original observations paired in the distance matrix. The behavior of this function is very similar to the MATLAB linkage function.

A $(n-1)$ by 4 matrix `Z` is returned. 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.

The following linkage methods are used to compute the distance $d(s,t)$ between two clusters s and t. The algorithm begins with a forest of clusters that have yet to be used in the hierarchy being formed.



File Edit View History Bookmarks Tools Help
scipy.cluster.hierarchy.linkage –
https://docs.scipy.org/doc/scipy/reference/generated/scipy.cluster.hierarchy.linkage
Getting started
User Guide
API reference
Development
Release notes
1.9.3 (stable)
Search the docs ...
Clustering package (scipy.cluster)
K-means clustering and vector quantization (scipy.cluster.vq)
Hierarchical clustering (scipy.cluster.hierarch
Constants (scipy.constants)
Discrete Fourier

vectors; ignored otherwise. See the `pdist` function for a list of valid distance metrics. A custom distance function can also be used.

optimal_ordering : ***bool, optional***

If True, the linkage matrix will be reordered so that the distance between successive leaves is minimal. This results in a more intuitive tree structure when the data are visualized. defaults to False, because this algorithm can be slow, particularly on large datasets [2]. See also the `optimal_leaf_ordering` function.

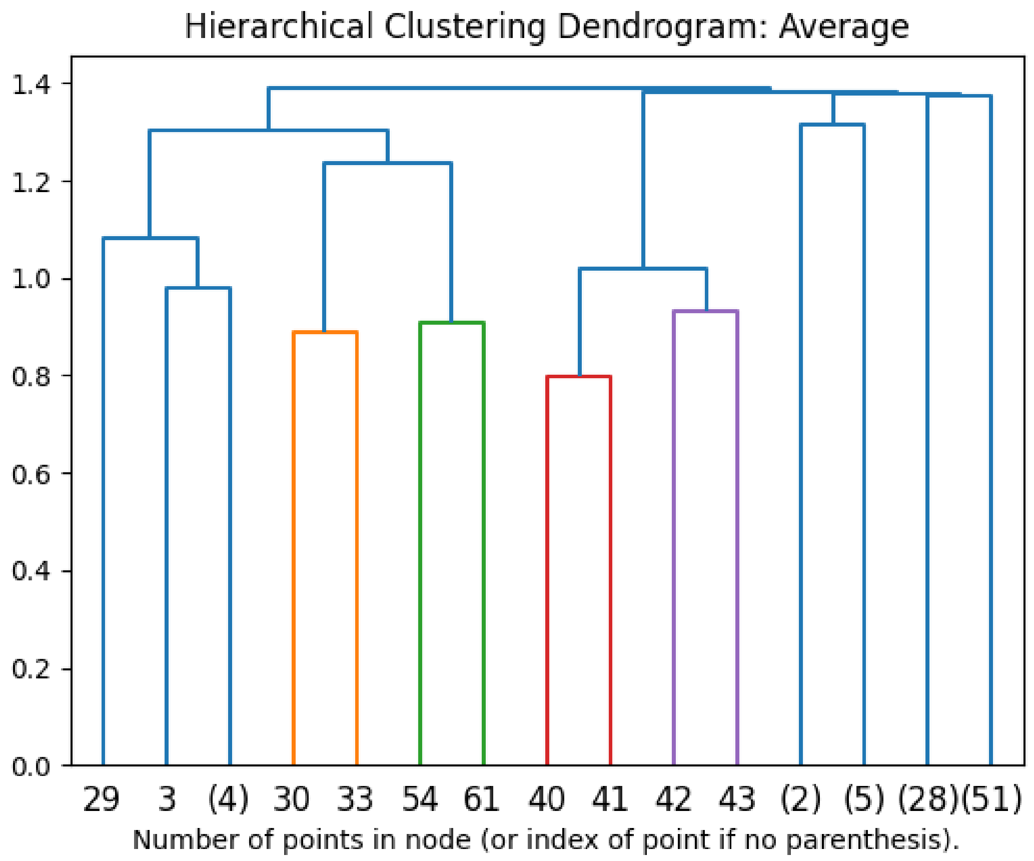
New in version 1.0.0.

Returns: **Z** : ***ndarray***

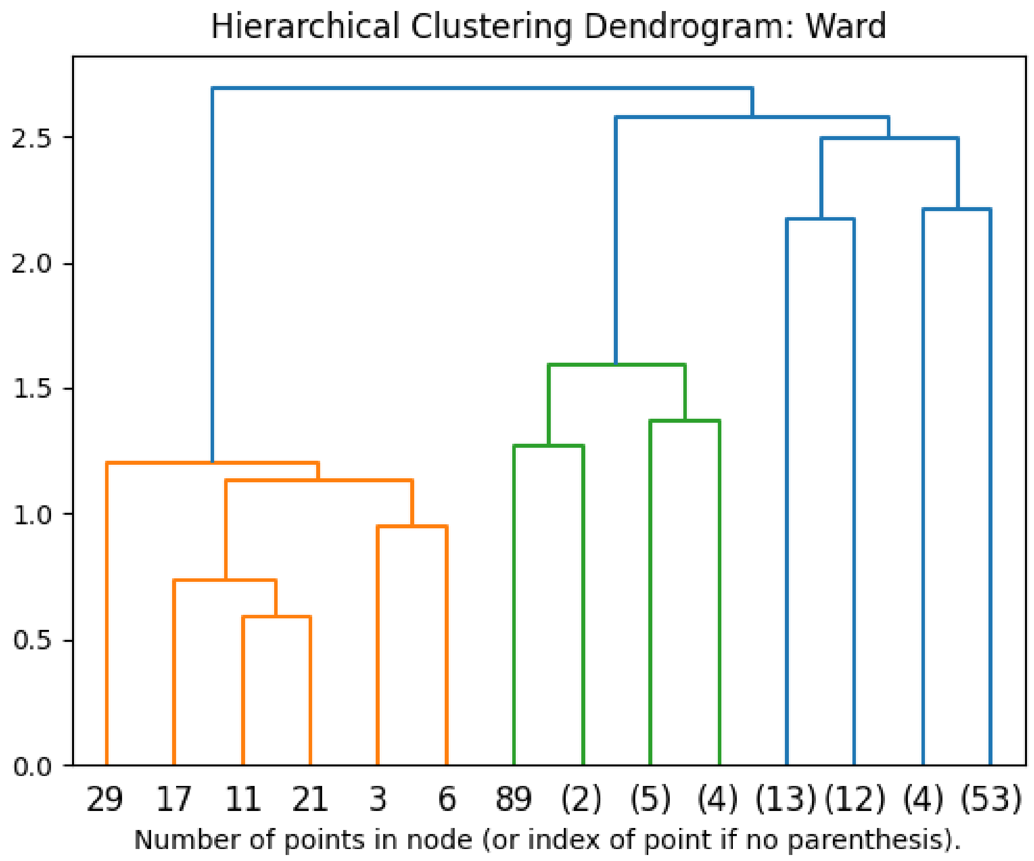
The hierarchical clustering encoded as a linkage matrix.

```
from scipy.cluster.hierarchy import dendrogram

def plot_dendrogram(model, **kwargs):
   # Create linkage matrix and then plot the dendrogram
   # create the counts of samples under each node
   counts = np.zeros(model.children_.shape[0])
   n_samples = len(model.labels_)
   for i, merge in enumerate(model.children_):
      current_count = 0
      for child_idx in merge:
         if child_idx < n_samples:
            current_count += 1  # leaf node
         else:
            current_count += counts[child_idx - n_samples]
      counts[i] = current_count

   linkage_matrix = np.column_stack(
      [model.children_, model.distances_, counts]
   ).astype(float)

   # Plot the corresponding dendrogram
   dendrogram(linkage_matrix, **kwargs)
```

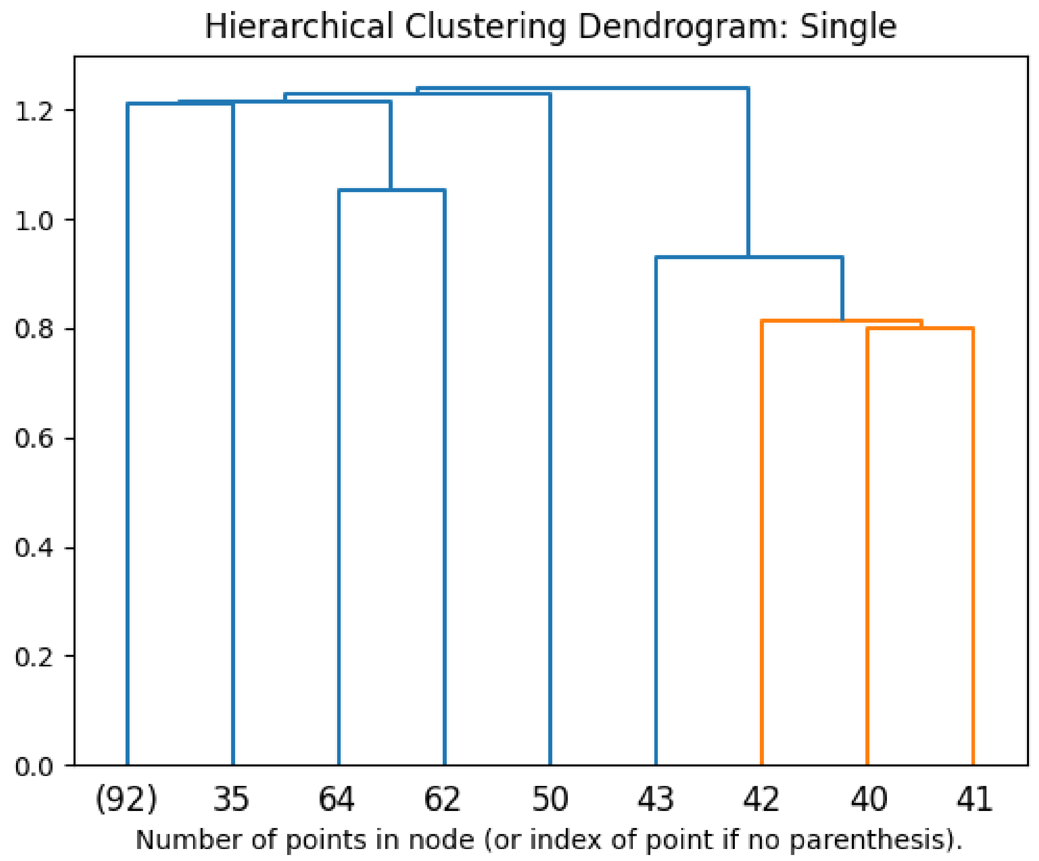
```
f = open("bible.txt", "r")
docs = f.readlines()
f.close()
N = 100
firstk = docs[0:N]

cv = TfidfVectorizer(max_df=0.4, min_df=2)
X = cv.fit_transform(firstk).toarray()

# setting distance_threshold=0 ensures we compute the full tree.
model = AgglomerativeClustering(distance_threshold=0, n_clusters=None,
linkage='average')

model = model.fit(X)
plt.title("Hierarchical Clustering Dendrogram: Average")
# plot the top three levels of the dendrogram
plot_dendrogram(model, truncate_mode="level", p=3)
plt.xlabel("Number of points in node (or index of point if no parenthesis).")
plt.show()
```

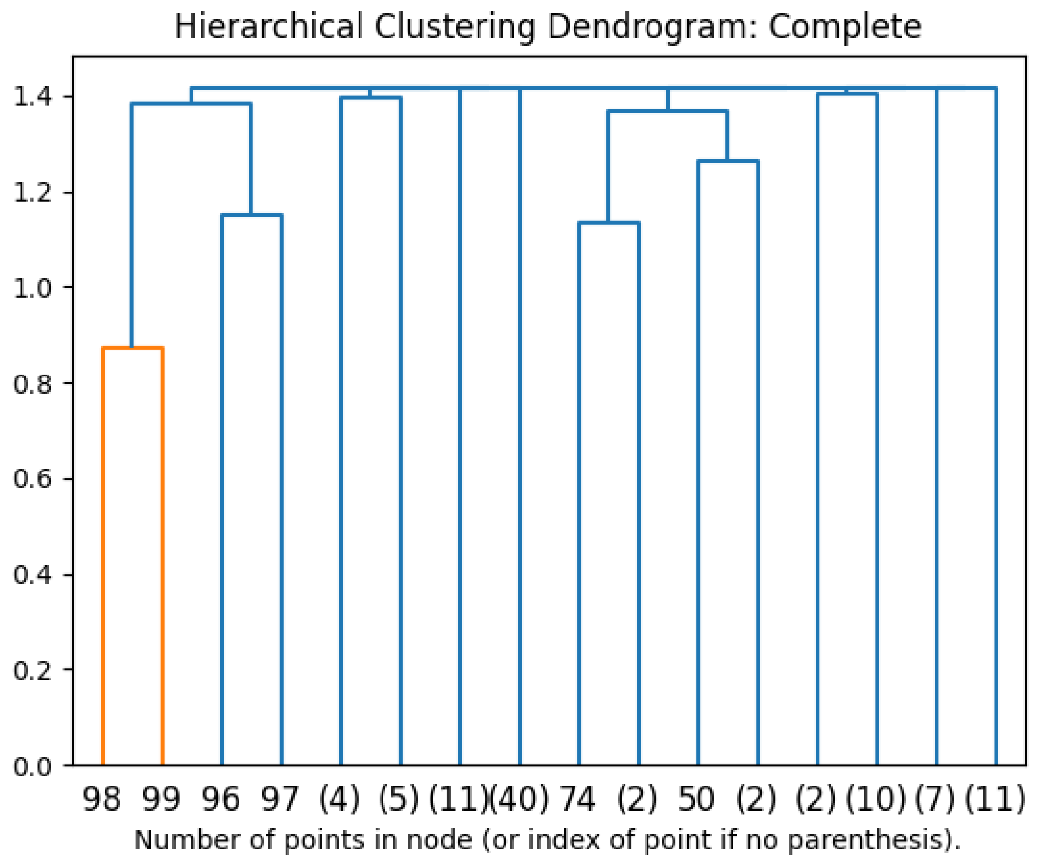


Hierarchical Clustering Dendrogram: Average
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
29 3 (4) 30 33 54 61 40 41 42 43 (2) (5) (28)(51)
Number of points in node (or index of point if no parenthesis).



Hierarchical Clustering Dendrogram: Ward
2.5
2.0
1.5
1.0
0.5
0.0
29 17 11 21 3 6 89 (2) (5) (4) (13) (12) (4) (53)
Number of points in node (or index of point if no parenthesis).


Hierarchical Clustering Dendrogram: Single
1.2
1.0
0.8
0.6
0.4
0.2
0.0
(92)
35
64
62
50
43
42
40
41
Number of points in node (or index of point if no parenthesis).


Hierarchical Clustering Dendrogram: Complete
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
98
99
96
97
(4)
(5)
(11)
(40)
74
(2)
50
(2)
(2)
(10)
(7)
(11)
Number of points in node (or index of point if no parenthesis).

17.5 Optimality of HAC

To state the optimality conditions of hierarchical clustering precisely, we first define the combination similarity COMB-SIM of a clustering $\Omega = \{\omega_1, \ldots, \omega_K\}$ as the smallest combination similarity of any of its K clusters:

$$\text{COMB-SIM}(\{\omega_1, \ldots, \omega_K\}) = \min_k \text{COMB-SIM}(\omega_k)$$

Recall that the combination similarity of a cluster ω that was created as the merge of ω_1 and ω_2 is the similarity of ω_1 and ω_2 (page 378).

'ERING We then define $\Omega = \{\omega_1, \ldots, \omega_K\}$ to be *optimal* if all clusterings Ω' with k clusters, $k \leq K$, have lower combination similarities:

$$|\Omega'| \leq |\Omega| \Rightarrow \text{COMB-SIM}(\Omega') \leq \text{COMB-SIM}(\Omega)$$

single-link The combination similarity of a cluster ω is the smallest similarity of any bipartition of the cluster, where the similarity of a bipartition is the largest similarity between any two documents from the two parts:

$$\text{COMB-SIM}(\omega) = \min_{\{\omega' : \omega' \subset \omega\}} \max_{d_i \in \omega'} \max_{d_j \in \omega - \omega'} \text{SIM}(d_i, d_j)$$

where each $\langle \omega', \omega - \omega' \rangle$ is a bipartition of ω.

complete-link The combination similarity of a cluster ω is the smallest similarity of any two points in ω: $\min_{d_i \in \omega} \min_{d_j \in \omega} \text{SIM}(d_i, d_j)$.

GAAC The combination similarity of a cluster ω is the average of all pairwise similarities in ω (where self-similarities are not included in the average): Equation (17.3).

Exercise 17.4

Show the equivalence of the two definitions of combination similarity: the process definition on page 378 and the static definition on page 393.

method	combination similarity	time compl.	optimal?	comment
single-link	max inter-similarity of any 2 docs	$\Theta(N^2)$	yes	chaining effect
complete-link	min inter-similarity of any 2 docs	$\Theta(N^2 \log N)$	no	sensitive to outliers
group-average	average of all sims	$\Theta(N^2 \log N)$	no	best choice for most applications

▶ **Table 17.1** Comparison of HAC algorithms.