Introduction to Data Mining

Hierarchical Clustering

CPSC/AMTH 445a/545a

Guy Wolf
guy.wolf@yale.edu

Yale University
Fall 2016
Outline

1. Hierarchical clustering
   - Divisive & agglomerative approaches
   - Dendrogram visualization
   - Bisecting $k$-means

2. Agglomerative clustering
   - Single linkage
   - Complete linkage
   - Average linkage
   - Ward’s method

3. Large-scale clustering
   - CURE
   - BIRCH
   - Chameleon
Question: how many cluster should we find in the data?
Question: how many cluster should we find in the data?

Six Clusters
Question: how many clusters should we find in the data?
**Question:** how many clusters should we find in the data?

**Two Clusters**
**Question:** how many clusters should we find in the data?

**Suggestion:** why not consider all options in a single hierarchy?
A hierarchical approach can be useful when considering versatile cluster shapes:

2-means

By first detecting many small clusters, and then merging them, we can uncover patterns that are challenging for partitional methods.
Hierarchical clustering

A hierarchical approach can be useful when considering versatile cluster shapes:

10-means

By first detecting many small clusters, and then merging them, we can uncover patterns that are challenging for partitional methods.
Hierarchical clustering methods produce a set of nested clusters organized in a hierarchy tree.

- The cluster hierarchy is typically **visualized using dendrograms**
- Such approaches are applied either to provide multiresolution data organization, or to alleviate computational challenges when clustering big datasets
- In general, two approaches are applied to build nested clusters: divisive clustering and agglomerative clustering.

**Divisive approaches** start with the entire data as one cluster, and then iteratively split “loose” clusters until a stopping criterion (e.g., $k$ clusters or tight enough clusters) is satisfied.

**Agglomerative approaches** start with small tight clusters, or even with single-point clusters, and then iteratively merge close clusters until only a single one remains.
Hierarchical clustering

Dendrogram visualization

A dendrogram is a tree graph that visualizes a sequence of cluster merges or divisions:

![Dendrogram Image]

**Divisive** methods take a top-down root $\rightarrow$ leaves approach. **Agglomerative** ones take a bottom-up leaves $\rightarrow$ root approach.
Hierarchical clustering

Bisecting k-means

Bisecting $k$-means is a divisive algorithm that utilized the $k$-means iteratively to bisect the data into clusters.

Bisecting $k$-means

Use 2-means to split the data into two clusters¹

While there are less than $k$ clusters:

- Select $C$ as the cluster with the highest $SSE$
- Use 2-means to split $C$ into two clusters¹
- Replace $C$ with the two new clusters

The hierarchical approach in this case is used to stabilize some of the weaknesses of the original $k$-means algorithm, and not for data organization purposes.

¹Choose best $SSE$ out of $t$ attempts
Hierarchical clustering

Bisecting k-means

Example

Iteration 1
Hierarchical clustering

Bisecting k-means

Example

![Iteration 2](image)
Hierarchical clustering

Bisecting k-means

Example

Iteration 3
Hierarchical clustering
Bisecting k-means

Example

Iteration 4
Hierarchical clustering

Bisecting k-means

Example

Iteration 5
Hierarchical clustering

Bisecting k-means

Example

Iteration 6

Y

0

-2

-4

-6

0

5

10

15

20

X
Hierarchical clustering

Example

Iteration 7
Hierarchical clustering

Bisecting k-means

Example
Hierarchical clustering

Bisecting k-means

Example

Iteration 9
Hierarchical clustering
Bisecting k-means

Example

Iteration 10

- 8
- 6
- 4
- 2
 0
- 2
- 4
- 6
- 8
0 5 10 15 20

Y
X
Agglomerative clustering approaches are more popular than divisive ones. They all use variations of the following simple algorithm:

**Agglomerative clustering paradigm**

**Build** a singleton cluster for each data point  
**Repeat** the following steps:  
- **Find** the two closest clusters  
- **Merge** these two clusters together  
**Until** there is only a single cluster

Two main choices distinguish agglomerative clustering algorithms:

1. How to quantify proximity between clusters  
2. How to merge clusters and efficiently update this proximity
Agglomerative clustering approaches are more popular than divisive ones. They all use variations of the following simple algorithm:

**Agglomerative clustering paradigm**

**Build** a singleton cluster for each data point

**Repeat** the following steps:
- **Find** the two closest clusters
- **Merge** these two clusters together

**Until** there is only a single cluster

With proper implementation, this approach is also helpful for Big Data processing, since each iteration considers a smaller coarse-grained version of the dataset.
Agglomerative clustering

Proximity Matrix

<table>
<thead>
<tr>
<th></th>
<th>p1</th>
<th>p2</th>
<th>p3</th>
<th>p4</th>
<th>p5</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Agglomerative clustering

Hierarchical Clustering

Proximity Matrix

C1 C2 C3 C4 C5
C1
C2
C3
C4
C5

C1
C2
C3
C4
C5

p1
p2
p3
p4
p9
p10
p11
p12
Agglomerative clustering

Proximity Matrix

C1  C5  C3  C4
C1   ?
C2 U C5  ?  ?  ?  ?
C3   ?
C4   ?

C2 U C5

C1  C5
C2  ?
C3  ?
C4  ?

p1  p2
p3  p4
p9  p10  p11  p12
How to quantify distance or similarity between clusters?

Suggestion #1: represent clusters by centroids and use distance/similarity between them.

Problem: this approach ignores the shapes of the clusters.

Suggestion #2: combine pairwise distances between each point in one cluster and each point in the other cluster. This approach is called linkage.
How to quantify distance or similarity between clusters?

**Suggestion #1:** represent clusters by centroids and use distance/similarity between them.
How to quantify distance or similarity between clusters?

**Suggestion #1:** represent clusters by centroids and use distance/similarity between them.

**Problem:** this approach ignores the shapes of the clusters.
Agglomerative clustering

Linkage

How to quantify distance or similarity between clusters?

**Suggestion #1:** represent clusters by centroids and use distance/similarity between them.

**Problem:** this approach ignores the shapes of the clusters.

**Suggestion #2:** combine pairwise distances between each point in one cluster and each point in the other cluster. This approach is called **linkage**.
**Single linkage** uses minimal distance (or maximum similarity) between a point in one cluster and a point in the other cluster.

Only one inter-cluster link determines the distance, while many other links can be significantly weaker.
Agglomerative clustering

Single linkage

Example

Hierarchical Clustering

Yale - Fall 2016 9 / 17
**Complete linkage** uses maximal distance (or minimal similarity) between a point in one cluster and a point in the other cluster.

In some sense, all inter-cluster links are considered since they must all be strong to have a small distance.
Agglomerative clustering

Complete linkage

Example
**Average linkage** uses mean distance (or similarity) between points in one cluster and points in the other cluster.

Less susceptible than single- and complete-linkage to noise and outliers, but biased toward globular clusters.
Agglomerative clustering

Average linkage

Example

Hierarchical Clustering

Yale - Fall 2016
Instead of considering connectivity between clusters, we can also consider the impact of merging clusters on their quality.

**Ward’s method** compares the total $SSE$ of the two clusters to the $SSE$ of a single cluster obtained by merging them.

- Similar to average-linkage with squared distances as dissimilarities.
- Like average-linkage, biased toward globular clusters while being somewhat stable to noise and outliers.

Ward’s method provided an agglomerative/hierarchical analogue to $k$-means.
Large-scale clustering

Hierarchical clustering is not only useful for data organization, but also for large scale data processing, even without special interpretability. A common approach for clustering big data is to iteratively coarse-grain the data to reduce its size, until a desired resolution (e.g., number or size of clusters) is reached. Each coarse-graining iteration is achieved finding (and merging) small tight clusters.

Two of the main challenges in implementing such approaches are:

1. Finding a compact representation of clusters that allows merging and comparisons,
2. Efficient data scanning strategy during the initial cluster construction process and coarse-graining iterations.

Such methods also apply various advanced implementation techniques that are beyond the scope of this course.
CURE (Clustering Using REpresentatives) extends the idea of \( k \)-means by choosing a small set of \( r \) points to represent the cluster instead of a single centroid point.

Given a cluster, CURE chooses a set of representative points \( \{x_1, \ldots, x_r\} \) using the following steps:

- Compute the centroid \( \hat{c} \) of the cluster.
- Set \( x_1 \) to be the farthest point in the cluster from \( \hat{c} \).
- For \( i = 2, \ldots, r \), set \( x_i \) to be the farthest point from all previous representatives \( x_1, \ldots, x_{i-1} \).

Notice that these representatives are aimed to capture the borders of the cluster rather than its center, unlike \( k \)-means.
Using border points as cluster representatives allows CURE to capture non-globular and concave-shaped clusters.

However, the farthest-point selection is sensitive to noise and outliers, so CURE “shrinks” these points toward the cluster center.

Each representative $x_i$ is replaced with $\hat{x}_i = x_i - \alpha (x_i - \hat{c})$. Since this shrinkage is relative to the distance, outliers are more affected by it than other points.

The shrinkage factor $\alpha$ controls the correction magnitude, and setting $\alpha = 1$ gives the classic centroid-based cluster representation.
Using the cluster representatives, CURE applies a **single-link agglomerative clustering** approach, based on the minimal distance between representatives.

Additionally, instead of clustering the entire data in at once, CURE **partitions the dataset** into smaller local partitions. Then, the agglomerative clustering is applied to each of them (e.g., in parallel). Finally, the **coarse-grained data** from these clusterings are **merged together**, and agglomerative clustering is applied on this cluster collection.

The hierarchical approach in CURE is mainly aimed to cope with computational challenges, rather than find hierarchical data organization.
The full CURE algorithm also includes sampling and outlier-removal steps, as described in the following pipeline:

More details can be found in: **CURE: an efficient clustering algorithm for large databases** (*Guha, Rastogi, & Shim, 1998*).
Large-scale clustering

BIRCH

BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies), is an efficient centroid-based clustering method that aims to reduce memory-related overheads of the clustering process.

The main principle in BIRCH is to use a single scan of the data in order to produce tight clusters that can then be iteratively merged to form a cluster hierarchy.

To enable this scan, the algorithm requires a compressed in-memory data structure, with efficient amortized insertion time of each newly scanned data point.
The BIRCH algorithm introduces the notion of *Clustering Features* (CF) to compactly represent clusters:

**Cluster features**

Given a cluster \( C = \{\vec{x}_1, \ldots, \vec{x}_m\} \subseteq \mathbb{R}^n \), its cluster features are the triple \( CF = (m, \vec{LS}, \vec{SS}) \in \mathbb{R} \times \mathbb{R}^n \times \mathbb{R}^n \) where \( \vec{LS} = \sum_{i=1}^{m} \vec{x}_i \) and \( \vec{SS}[j] = \sum_{i=1}^{m}(\vec{x}_i[j])^2, j = 1, \ldots, n \).

Notice that given two disjoint clusters \( C_1 \) and \( C_2 \), their features are easily merged as \( CF_{1,2} = CF_1 + CF_2 \) for \( C_1 \cup C_2 \).
Large-scale clustering

BIRCH

Not only are the CF easily merged, but they also have sufficient information the computation of many important cluster properties, such as centroid, radius, diameter, and SSE.

**Examples**

**Centroid:** $\hat{c} = \frac{1}{m} \sum_{\vec{x} \in C} \vec{x} = \frac{1}{m} \vec{LS}$

**Radius:** consider $R^2 = \frac{1}{m} \sum_{\vec{x} \in C} \| \vec{x} - \hat{c} \|^2$, then $\sum_{\vec{x} \in C} \| \vec{x} - \hat{c} \|^2 = \sum_{\vec{x} \in C} \| \vec{x} \|^2 + m \| \hat{c} \|^2 - 2 \sum_{\vec{x} \in C} \langle \vec{x}, \hat{c} \rangle$, but then $\sum_{\vec{x} \in C} \langle \vec{x}, \hat{c} \rangle = \langle \sum_{\vec{x} \in C} \vec{x}, \hat{c} \rangle = m \| \hat{c} \|^2$ and $\sum_{\vec{x} \in C} \| \vec{x} \|^2 = \sum_{\vec{x} \in C} \sum_{j=1}^{n} (\vec{x}[j])^2 = \sum_{j=1}^{n} \vec{SS}[j]$, thus we get $R = \sqrt{\frac{1}{m} \| \vec{SS} \|_1 - \frac{1}{m^2} \| \vec{LS} \|_2^2}$
The clusters in BIRCH are built incrementally by scanning the dataset and inserting each data point to the closest cluster. These insertions amount to simple updates of the CF of the cluster.

However, the clusters are constrained to have a bounded diameter, and if no cluster can absorb a data point, the scan creates a new singleton cluster.

In order to enable efficient nearest-cluster searches, the CF’s are organized in a balanced CF-tree. The size of the tree is determined by technical considerations (e.g., memory size) in order to minimize paging and I/O overheads of the scanning process.
Each node in the tree is limited to have at most $B$ clusters.

The leaves of this tree hold tight clusters, while other node hold super-clusters, which also correspond to branches & child-nodes.
When a new point is scanned, the algorithm *recursively finds the closest CF* in each node (starting from the root) and follows the corresponding branch to *traverse the CF tree* until the closest CF is found in a leaf node.

Once a CF is found in a leaf node, the algorithm *checks whether it can absorb the data point* under the bounded diameter constraint. If a data point is absorbed by a cluster, its *CF is updated* accordingly, otherwise a *new CF is created* in the leaf node.

If the leaf node now has *more than $B$ clusters*, it is *split in two*, and the CF entries in its parent node are updated to replace the CF entry of the removed branch with two CF entries for the added branches.
Once a CF is found in a leaf node, the algorithm **checks whether it can absorb the data point** under the bounded diameter constraint. If a data point is absorbed by a cluster, its **CF is updated** accordingly, otherwise a **new CF is created** in the leaf node.

If the leaf node now has **more than B clusters**, it is **split in two**, and the CF entries in its parent node are updated to replace the CF entry of the removed branch with two CF entries for the added branches.

In any case, each update also **triggers updates in all the ancestor nodes** on the path toward the root of the CF tree. Similar to the leaf node update, these **updates may cause some internal nodes to be split**. If the root splits, it becomes an internal node and a new root is created.
Large-scale clustering

BIRCH

The full BIRCH algorithm uses the following steps:

- **Phase 1:** Load into memory by building a CF tree
- **Initial CF tree**
- **Phase 2 (optional):** Condense into desirable range by building a smaller CF tree
- **smaller CF tree**
- **Phase 3:** Global Clustering
- **Good Clusters**
- **Phase 4:** (optional and offline) Cluster Refining
- **Better Clusters**

More details can be found in: **BIRCH: an efficient data clustering method for very large databases** (Zhang, Ramakrishnan, & Livny, 1996).
Large-scale clustering
Chameleon

Chameleon uses **graph partitioning** together with graph-oriented **agglomerative clustering** to enable efficient and robust clustering in big datasets. It follows three main phases:

**Preprocessing phase:** Chameleon start by computing a sparse $k$-NN graph to capture local relationships between data points.

- Notice that $k$-NN neighborhoods are more robust in variable-density data.
Large-scale clustering
Chameleon

**Partitioning phase:** multilevel graph partitioning is used to find many well-connected clusters in the data.
- The working assumption of the algorithm is that these should be subclusters of the true data clusters.

**Hierarchical phase:** agglomerative clustering is applied to iteratively merge (sub)clusters.
- Instead of using linkage, Chameleon considers interconnectivity and closeness between two clusters.
- Each iteration merges a pair of clusters with the highest relative interconnectivity and relative closeness.
**Interconnectivity** between clusters is defined as the sum of edge weights that cross from one cluster to another.

**Closeness** between clusters is defined as the average of these weights.

The **relative** version of these quantities is obtained by normalization with the average corresponding quantity measured over bisections that split each cluster into two equal-size parts.

More details can be found in: Chameleon: Hierarchical clustering using dynamic modeling (Karypis, Han, & Kumar, 1999).
Hierarchical clustering provides a multiscale data organization.

- Dendrograms are typically used for visualizing the recovered nested cluster structure.
- Agglomerative clustering is a popular approach to build cluster hierarchies, based on linkage or impact on a suitable cluster quality measure.

It is also useful as a coarse-graining tool for scalable data processing.

- CURE performs scalable clustering based on sampling, partitioning and representative selection.
- BIRCH uses an efficient CF tree construction to optimize memory handling overheads of the clustering process.
- Chameleon is based on sparse graph partition, and an alternative cluster proximity that combines closeness and interconnectivity.

Many different methods are available to improve & extend these principles in both general and application-specific settings.