Unsupervised Learning: Clustering

Some material adapted from slides by Andrew Moore, CMU

Yann LeCun on Unsupervised Learning

"Most of human and animal learning is *unsupervised learning*. If intelligence was a cake, unsupervised learning would be the cake, *supervised learning* would be the icing on the cake, and *reinforcement learning* would be the cherry on the cake. … We know how to make the icing and the cherry, but we don't know how to make the cake. We need to solve the unsupervised learning problem before we can even think of getting to true AI. And that's just an obstacle we know about. What about all the ones we don't know about?"

-- Yann LeCun*, on AlphaGo's success and AI, 2016

** Head of Facebook AI, NYU CS Proffessor*

Unsupervised Learning

- Supervised learning used labeled data pairs (x, y) to learn a function $f : X \rightarrow y$
- But, what if we don't have labels?
- No labels = **unsupervised learning**
- Only some points are labeled = **semi-supervised learning**
  - Getting labels is expensive, so we only get a few
- **Clustering** is the unsupervised grouping of data points based on similarity
- It can be used for **knowledge discovery**

Clustering algorithms

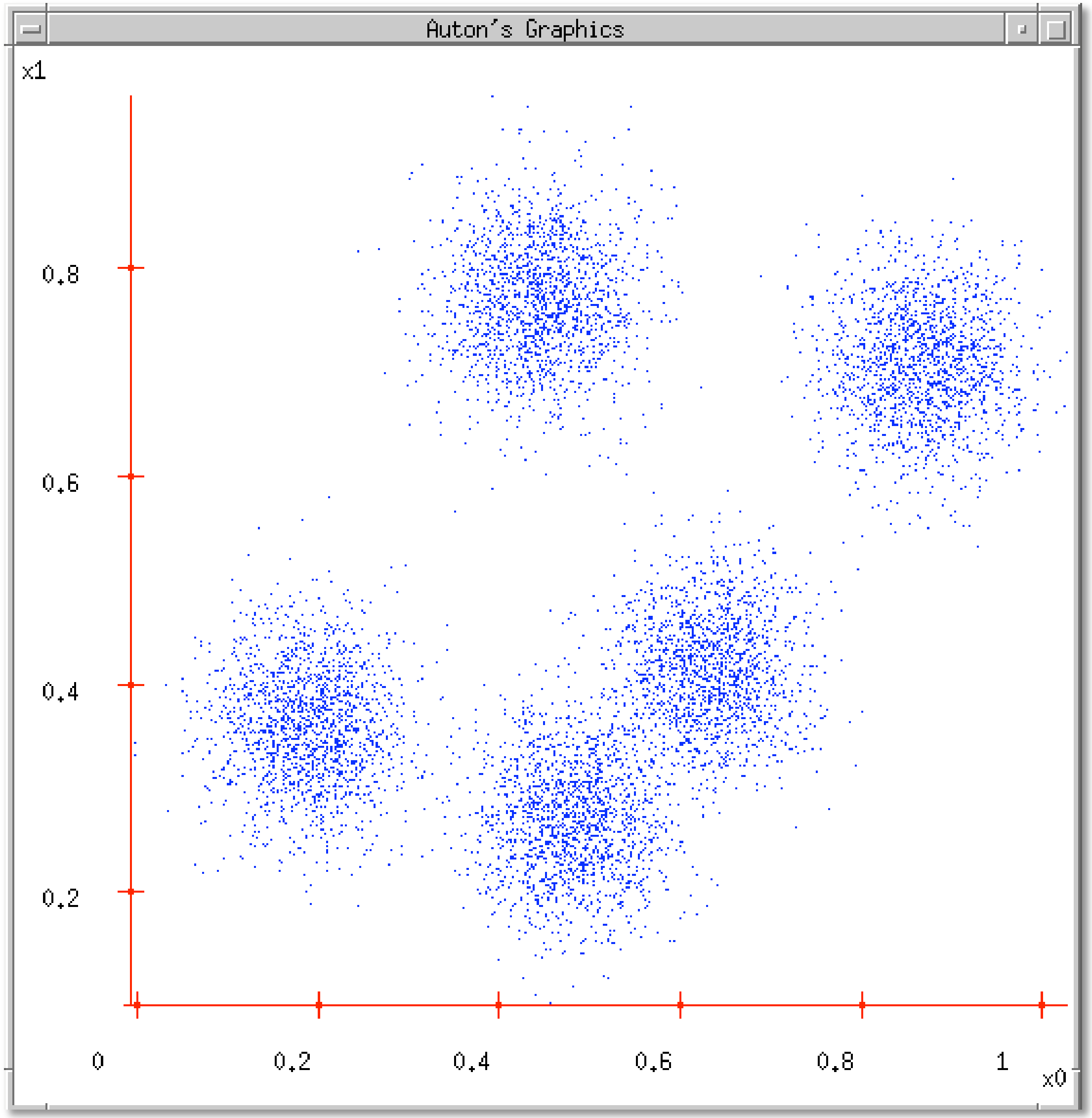
- Many clustering algorithms
- Clustering typically done using a **distance measure** defined between instances or points
- Distance defined by instance **feature space**, so it works with numeric features
  - Requires encoding of categorial values; may benefit from normalization
- We'll look at two popular approaches
  1. Centroid-based clustering
  2. Hierarchical clustering

Clustering Data

Given a collection of points (x,y), group them into one or more clusters based on their distance from one another

How many clusters are there?

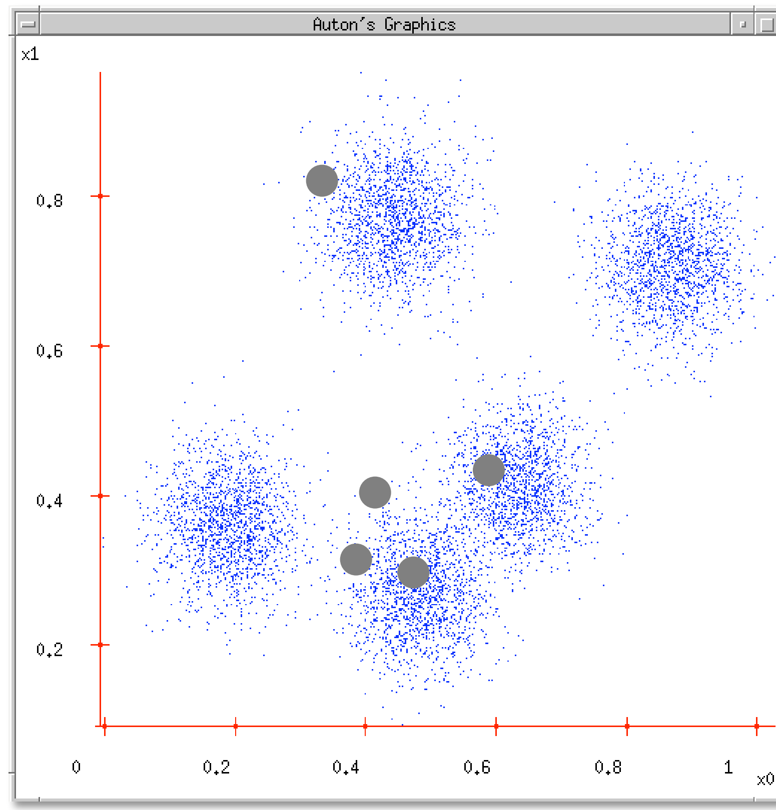
How can we find them



(1) K-Means Clustering

- Randomly choose k cluster center locations, aka **centroids**
- Loop until convergence
  - assign a point to cluster of the closest centroid
  - re-estimate cluster centroids based on its data assigned
- Convergence: no point is assigned to a different cluster

k = 5



distance, centroids

- Distance between points (X_0,Y_0,Z_0) and (X_1,Y_1,Z_1) is just $\mathrm{sqrt}((X_0- X_1)^2 + (Y_0- Y_1)^2 + (Z_0- Z_1)^2)$
- In numpy

```
>>> import numpy as np
>>> p1 = np.array([0,-2,0,1]) ; p2 = np.array([0,1,2,1]))
>>> np.linalg.norm(p1 - p2)
3.605551275463989
```

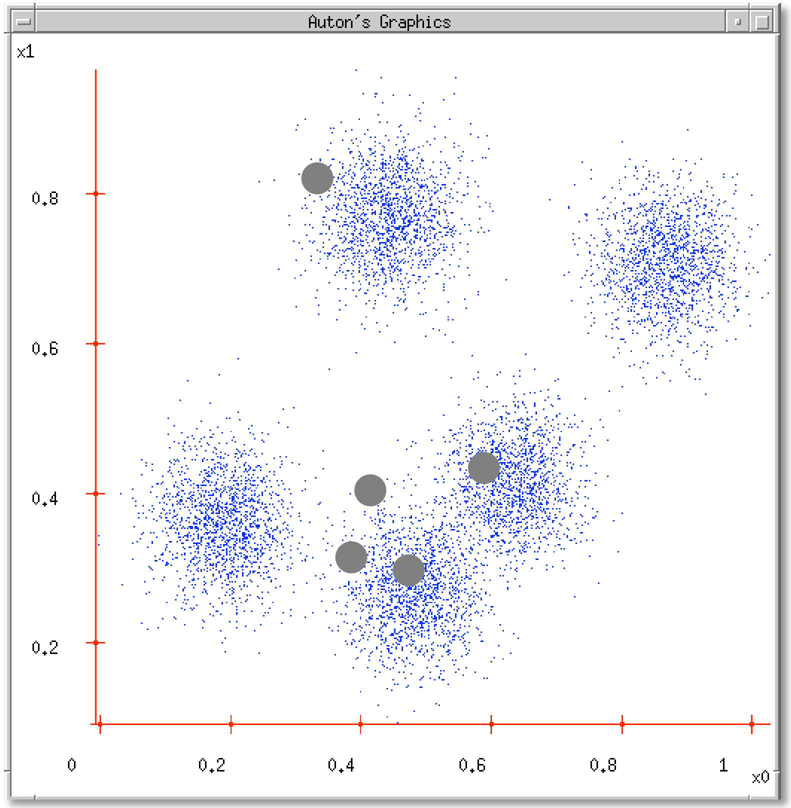
- Computing centroid of set of points easy

```
>>> points = np.array([[1,2,3], [2,1,1], [3,1,0]])    # 3D points
>>> centroid = np.mean(points, axis=0)        # get mean across columns
>>> centroid
array([2.0, 1.33, 1.33])
```

(1) K-Means Clustering

- Randomly choose k cluster center locations, aka **centroids**
- Loop until convergence
  - assign a point to cluster of the closest centroid
  - re-estimate cluster centroids based on its data assigned
- Convergence: no point is assigned to a different cluster

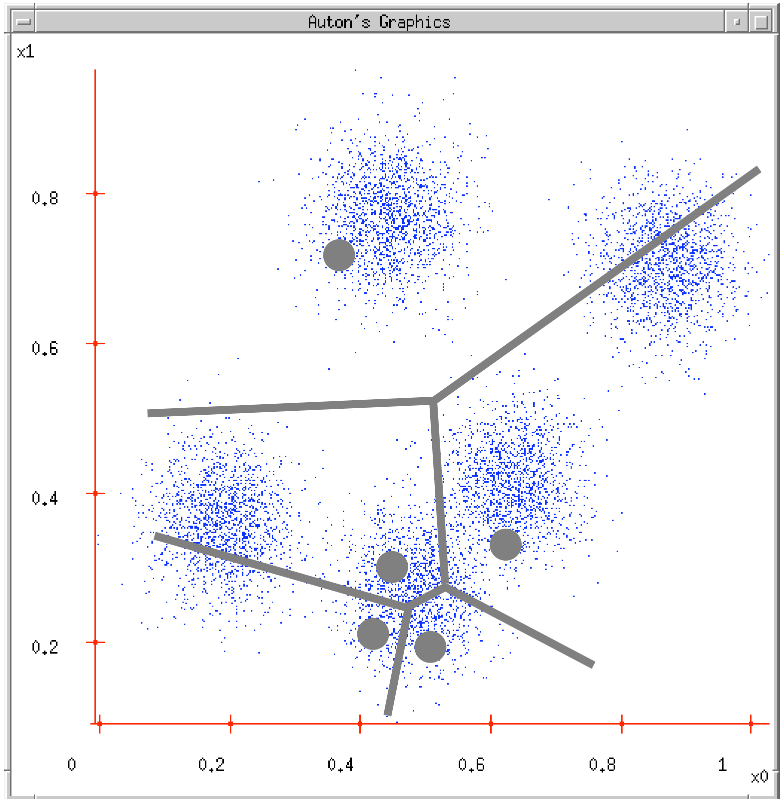
k = 5



K-Means Clustering

K-Means (k , data)

- Randomly choose k cluster center locations (centroids)
- Loop until convergence
  - Assign each point to the cluster of the closest centroid.
  - Re-estimate the cluster centroids based on the data assigned to each
- Convergence: no point is assigned to a different cluster

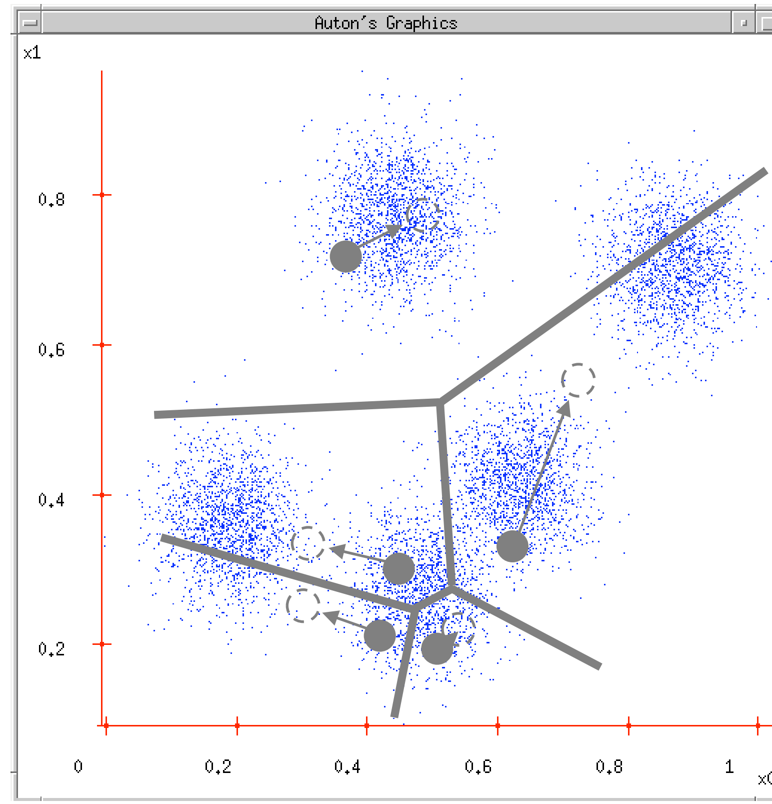


veroni diagram: add lines for regions of points closest to each centroid

K-Means Clustering

K-Means (k , data)

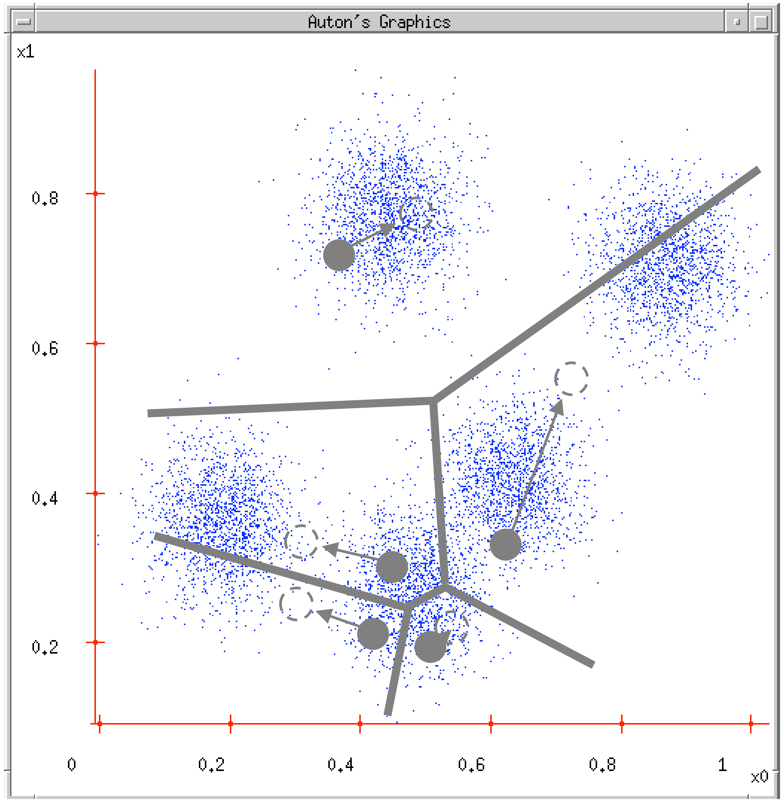
- Randomly choose k cluster center locations (centroids)
- Loop until convergence
  - Assign each point to the cluster of the closest centroid
  - Re-estimate the cluster centroids based on the data assigned to each
- Convergence: no point is assigned to a different cluster



K-Means Clustering

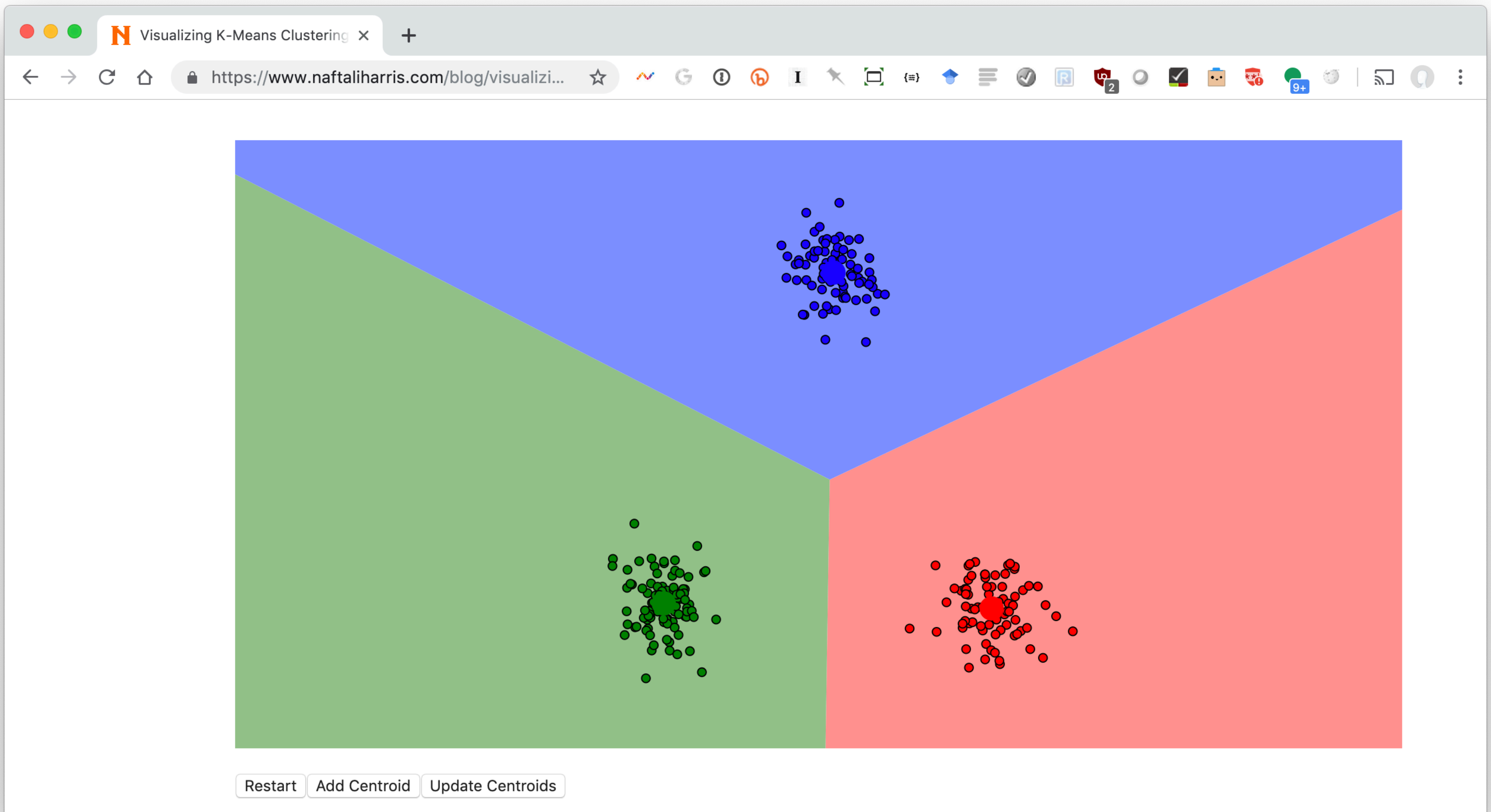
K-Means (k , data)

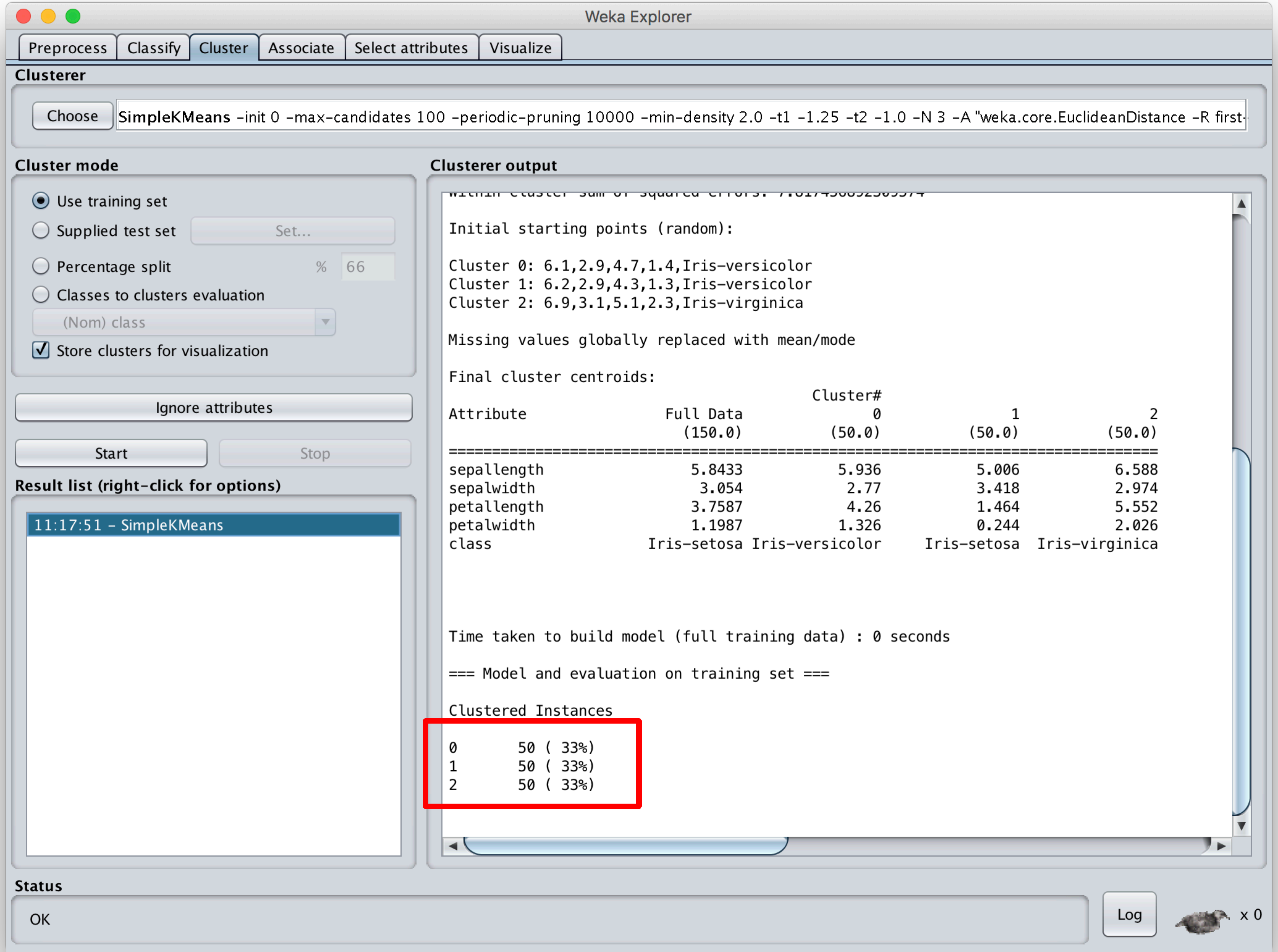
- Randomly choose k cluster center locations (centroids)
- Loop until convergence
  - Assign each point to the cluster of the closest centroid
  - Re-estimate the cluster centroids based on the data assigned to each
- Convergence: no point is assigned to a different cluster



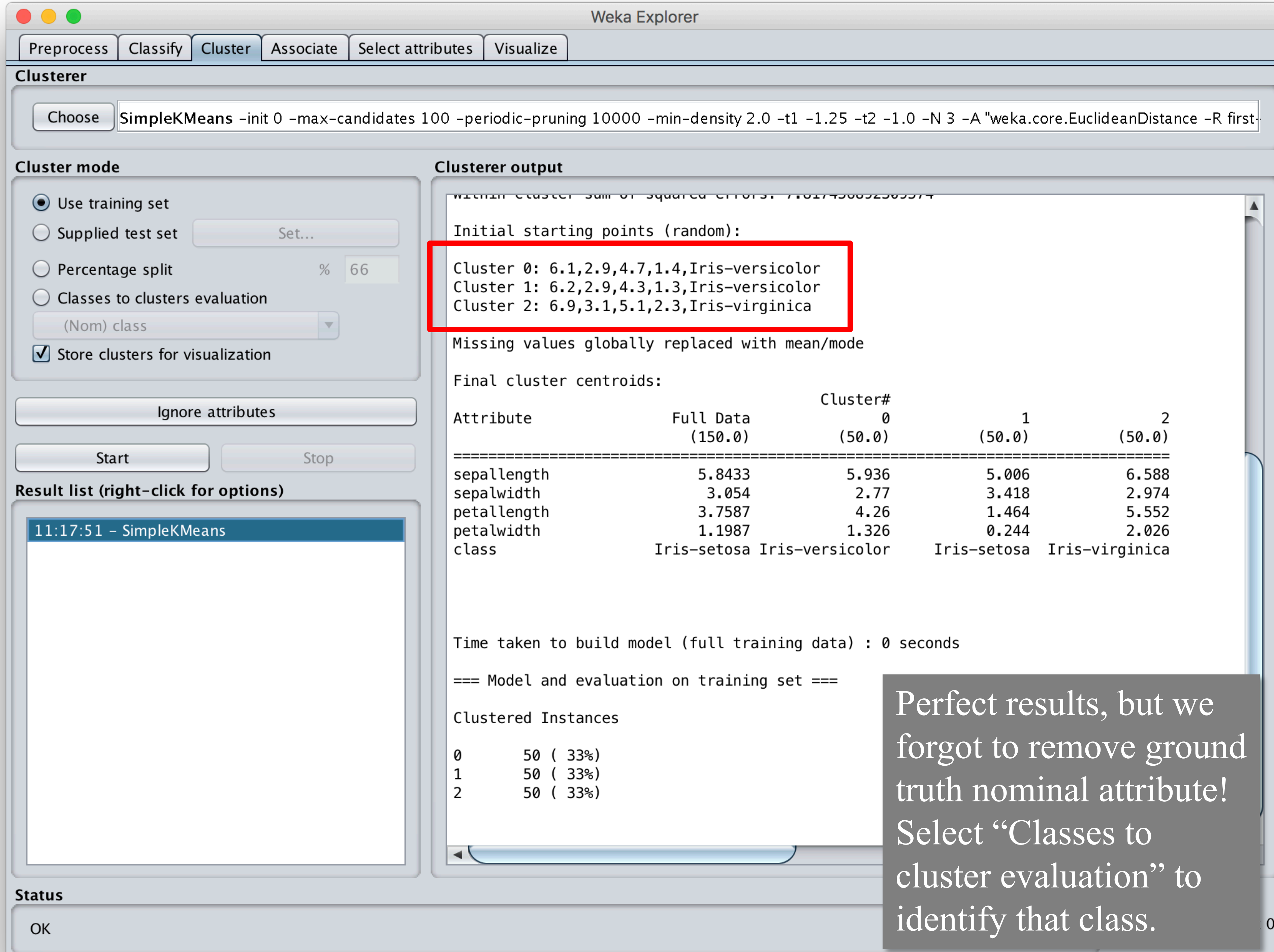
Visualizing k-means:

http://bit.ly/471kmean

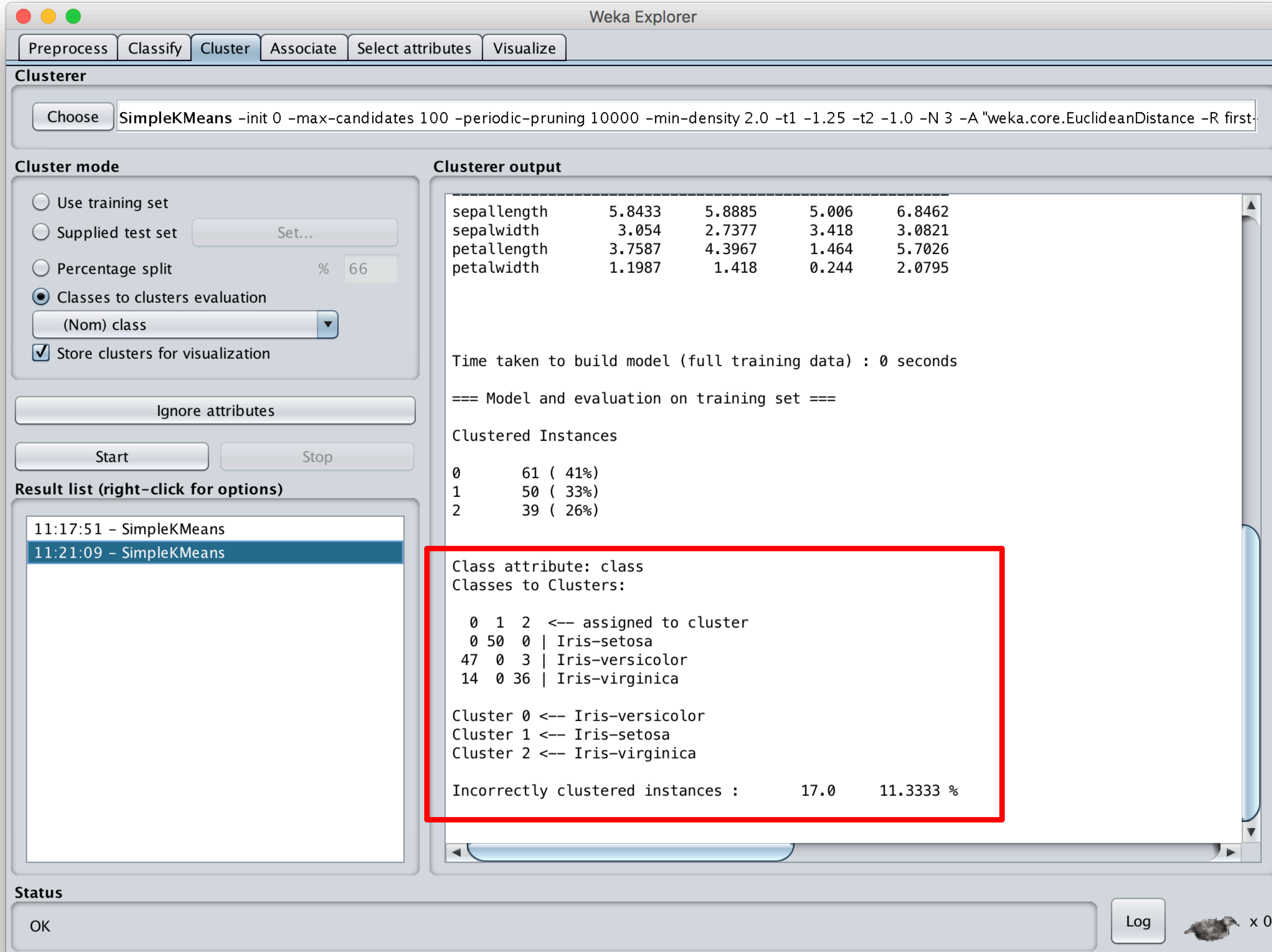




Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Clusterer
Choose
SimpleKMeans -init 0 -max-candidates 100 -periodic-pruning 10000 -min-density 2.0 -t1 -1.25 -t2 -1.0 -N 3 -A "weka.core.EuclideanDistance -R first-
Cluster mode
Use training set
Supplied test set
Set...
Percentage split
%
66
Classes to clusters evaluation
(Nom) class
Store clusters for visualization
Ignore attributes
Start
Stop
Result list (right-click for options)
11:17:51 - SimpleKMeans
Clusterer output
Initial starting points (random):
Cluster 0: 6.1,2.9,4.7,1.4,Iris-versicolor
Cluster 1: 6.2,2.9,4.3,1.3,Iris-versicolor
Cluster 2: 6.9,3.1,5.1,2.3,Iris-virginica
Missing values globally replaced with mean/mode
Final cluster centroids:
Cluster#
Attribute Full Data 0 1 2
(150.0) (50.0) (50.0) (50.0)
sepallength 5.8433 5.936 5.006 6.588
sepalwidth 3.054 2.77 3.418 2.974
petallength 3.7587 4.26 1.464 5.552
petalwidth 1.1987 1.326 0.244 2.026
class Iris-setosa Iris-versicolor Iris-setosa Iris-virginica
Time taken to build model (full training data) : 0 seconds
=== Model and evaluation on training set ===
Clustered Instances
0 50 (33%)
1 50 (33%)
2 50 (33%)
Status
OK
Log
x 0



Perfect results, but we forgot to remove ground truth nominal attribute! Select "Classes to cluster evaluation" to identify that class.



Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Clusterer
Choose
SimpleKMeans -init 0 -max-candidates 100 -periodic-pruning 10000 -min-density 2.0 -t1 -1.25 -t2 -1.0 -N 3 -A "weka.core.EuclideanDistance -R first-
Cluster mode
Use training set
Supplied test set
Set...
Percentage split
% 66
Classes to clusters evaluation
(Nom) class
Store clusters for visualization
Ignore attributes
Start
Stop
Result list (right-click for options)
11:17:51 - SimpleKMeans
11:21:09 - SimpleKMeans
Clusterer output
sepallength 5.8433 5.8885 5.006 6.8462
sepalwidth 3.054 2.7377 3.418 3.0821
petallength 3.7587 4.3967 1.464 5.7026
petalwidth 1.1987 1.418 0.244 2.0795
Time taken to build model (full training data) : 0 seconds
=== Model and evaluation on training set ===
Clustered Instances
0 61 (41%)
1 50 (33%)
2 39 (26%)
Class attribute: class
Classes to Clusters:
0 1 2 <-- assigned to cluster
0 50 0 | Iris-setosa
47 0 3 | Iris-versicolor
14 0 36 | Iris-virginica
Cluster 0 <-- Iris-versicolor
Cluster 1 <-- Iris-setosa
Cluster 2 <-- Iris-virginica
Incorrectly clustered instances : 17.0 11.3333 %
Status
OK
Log
x 0

scikit-learn v0.20.3

Other versions

Please **cite us** if you use the software.

2.3. Clustering

Clustering of unlabeled data can be performed with the module `sklearn.cluster`.

Each clustering algorithm comes in two variants: a class, that implements the `fit` method to learn the clusters on train data, and a function, that, given train data, returns an array of integer labels corresponding to the different clusters. For the class, the labels over the training data can be found in the `labels_` attribute.

Input data

One important thing to note is that the algorithms implemented in this module can take different kinds of matrix as input. All the methods accept standard data matrices of shape `[n_samples, n_features]`. These can be obtained from the classes in the `sklearn.feature_extraction` module. For `AffinityPropagation`, `SpectralClustering` and `DBSCAN` one can also input similarity matrices of shape `[n_samples, n_samples]`. These can be obtained from the functions in the `sklearn.metrics.pairwise` module.

2.3.1. Overview of clustering methods

07_IrisClustering.ipynb - Colab

https://colab.research.google.c...

07_IrisClustering.ipynb

File Edit View Insert Runtime Tools Help

COMMENT SHARE

CODE TEXT CELL CELL RAM Disk EDITING

sklearn K-means clustering on Fisher's Iris dataset

```
[14] %matplotlib inline

     from sklearn import cluster, datasets, metrics
     import numpy as np
     import matplotlib.pyplot as plt
     from mpl_toolkits.mplot3d import Axes3D
```

```
[15] # load training data
     iris = sklearn.datasets.load_iris()
```

The Iris dataset has 150 instances:

- X floats for Sepal Length, Sepal Width, Petal Length and Petal Width *y: integer (0,1,2) representing species (Setosa, Versicolour, Virginica)

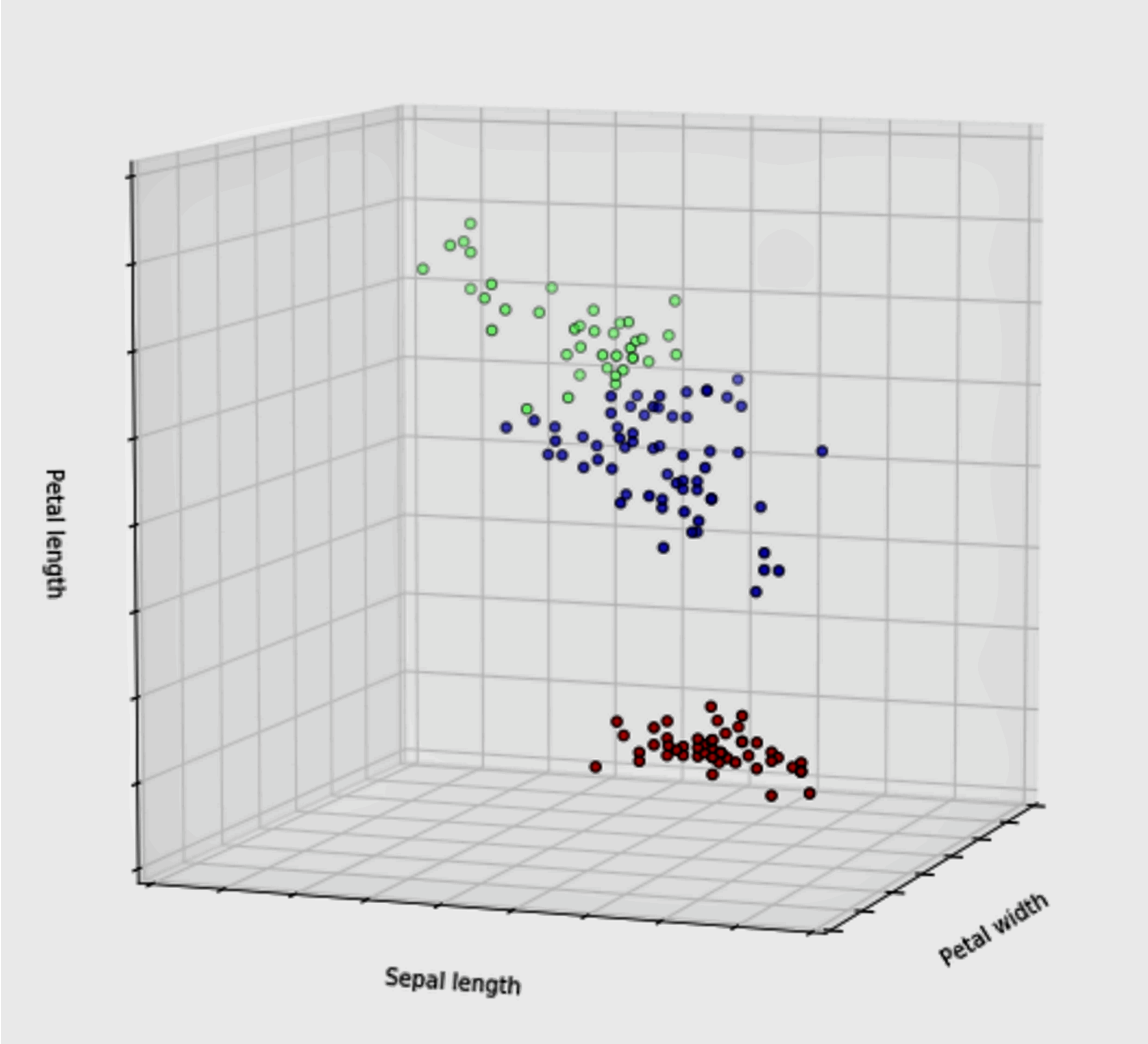
```
[20] X = iris.data
     y = iris.target
```

```
[25] # show first three rows of training instance data and the target classes
     print(X[:3])
     print(y[:3])
```

```
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]]
[0 0 0]
```

```
[27] # show all values for ground truth class (0,1,2)
     print(y)
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

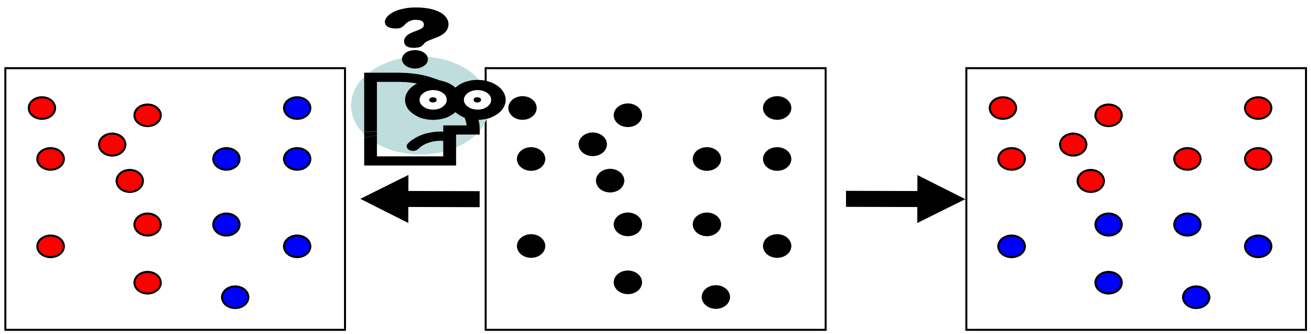

Petal length
Sepal length
Petal width

Problems with K-Means

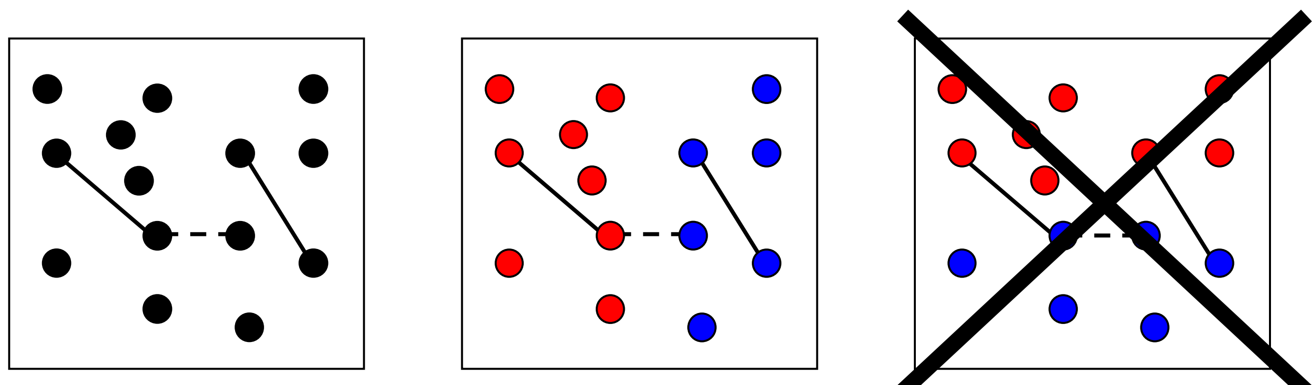
- Only works for numeric data (typically reals)
- ***Very*** sensitive to the initial points
  - Do many runs of k-Means, each with different initial centroids
  - Seed centroids using better method than random (e.g., **farthest-first** sampling)
- Must manually choose k
  - Learn optimal k for clustering
  - Note: requires a performance measure

Problems with K-Means

- How do you tell it which clustering you want?



- Constrained clustering technique



Same-cluster constraint (must-link)

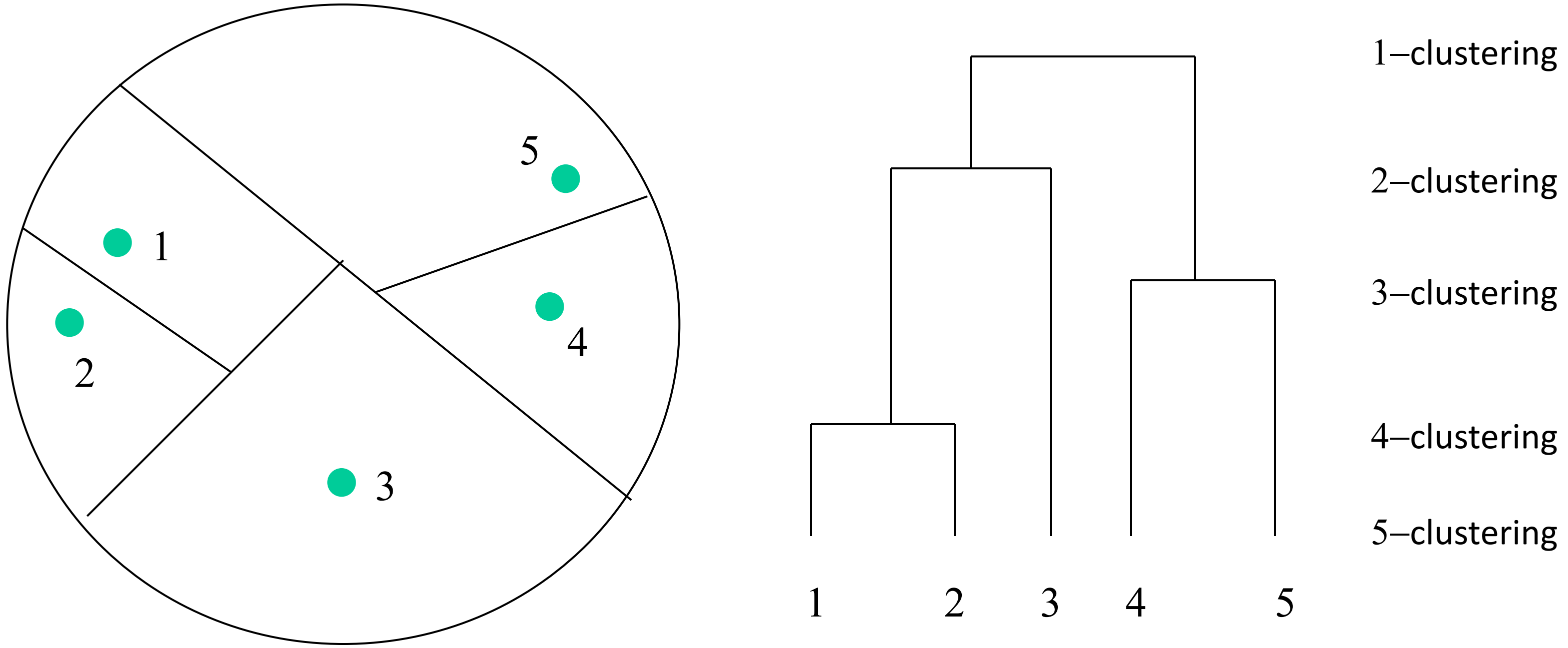
Different-cluster constraint (cannot-link)

(2) Hierarchical clustering

- **Agglomerative**
  - **bottom up** approach: elements start as individual clusters & clusters are merged as one moves up the hierarchy
- **Divisive**
  - **top down** approach: elements start as a single cluster & clusters are split as one moves down the hierarchy

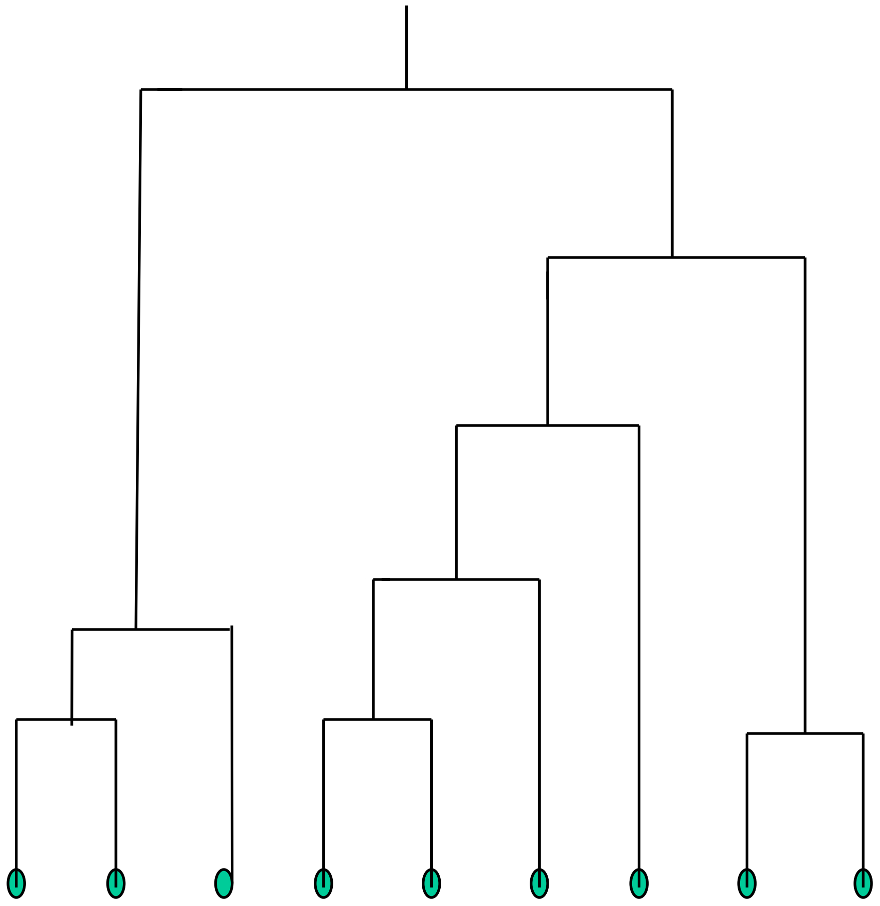
Hierarchical Clustering

Recursive partitioning/merging of a data set



Dendogram

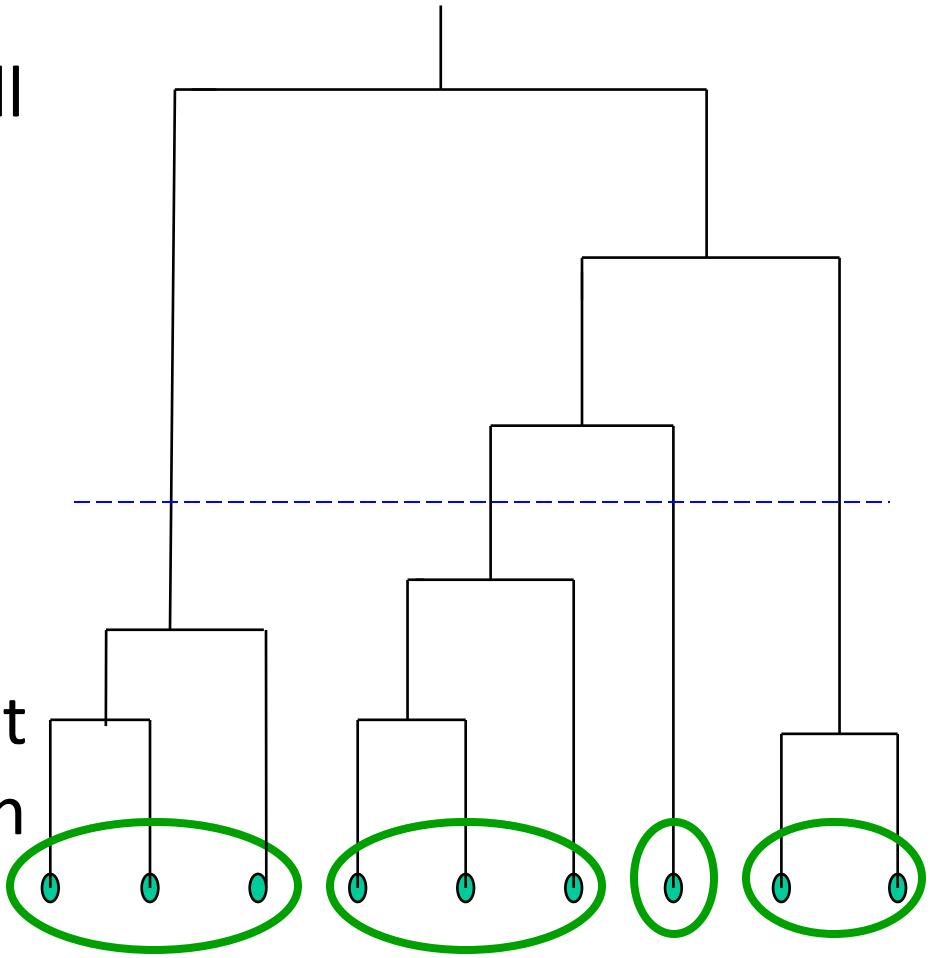
- Tree structure representing all data partitionings
- Constructed as clustering proceeds



Nine items

Dendogram

- Tree structure representing all data partitionings
- Constructed as clustering proceeds
- Get a K-clustering by looking at **connected** components at any given level
- Often binary dendograms, but n-ary ones easy to obtain with minor algorithm changes



Four clusters

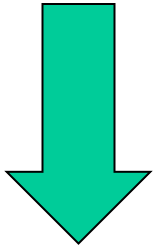
Hierarchical clustering advantages

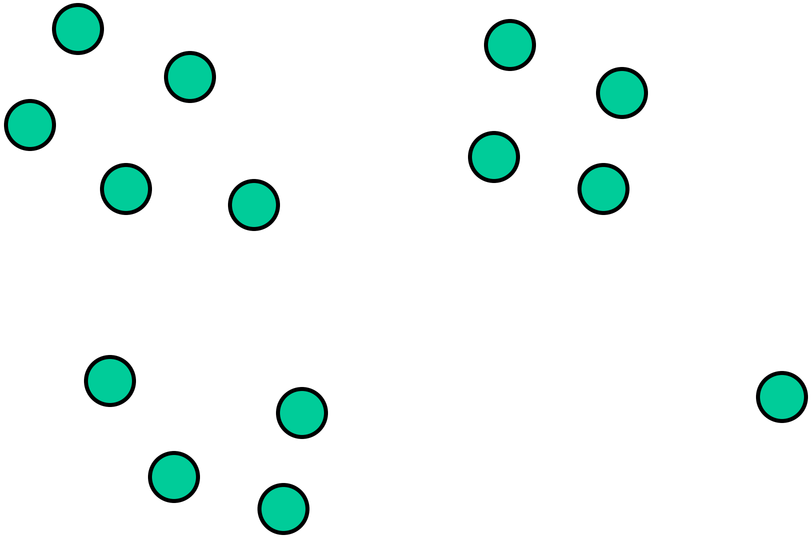
- Need not specify number of clusters
- Good for data visualization
  - See how data points interact at many levels
  - Can view data at multiple granularity levels
  - Understand how all points interact
- Specifies all of the K clusterings/partitions

Divisive hierarchical clustering

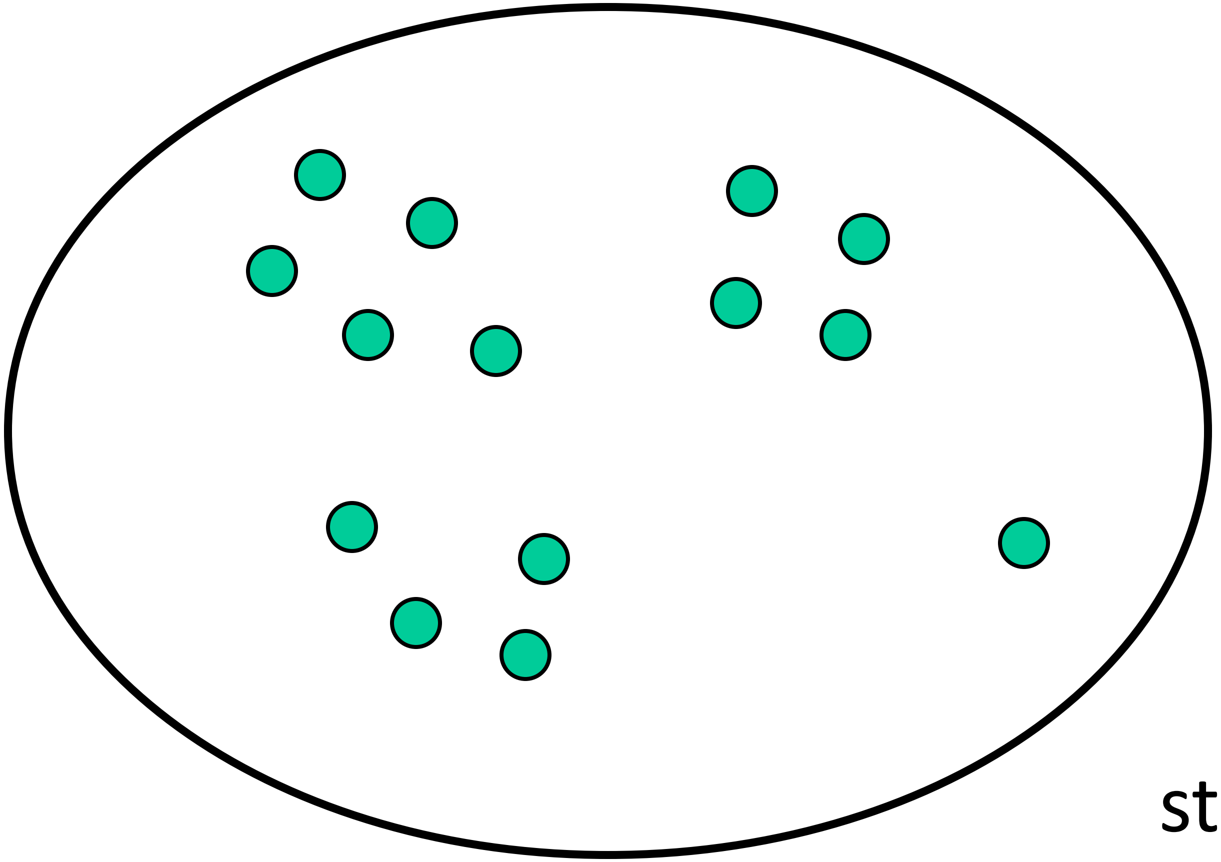
- Top-down
- Finding best partitioning of data generally exponential in time
- Common approach:
  - Let **C** be a set of clusters
  - Initialize **C** to be a one-clustering of data
  - While there exists a cluster *c* in **C**
    - remove *c* from **C**
    - partition *c* into 2 clusters (c_1 and c_2) using a flat clustering algorithm (e.g., k-means)
    - Add to c_1 and c_2 **C**
- Bisecting k-means

Divisive clustering



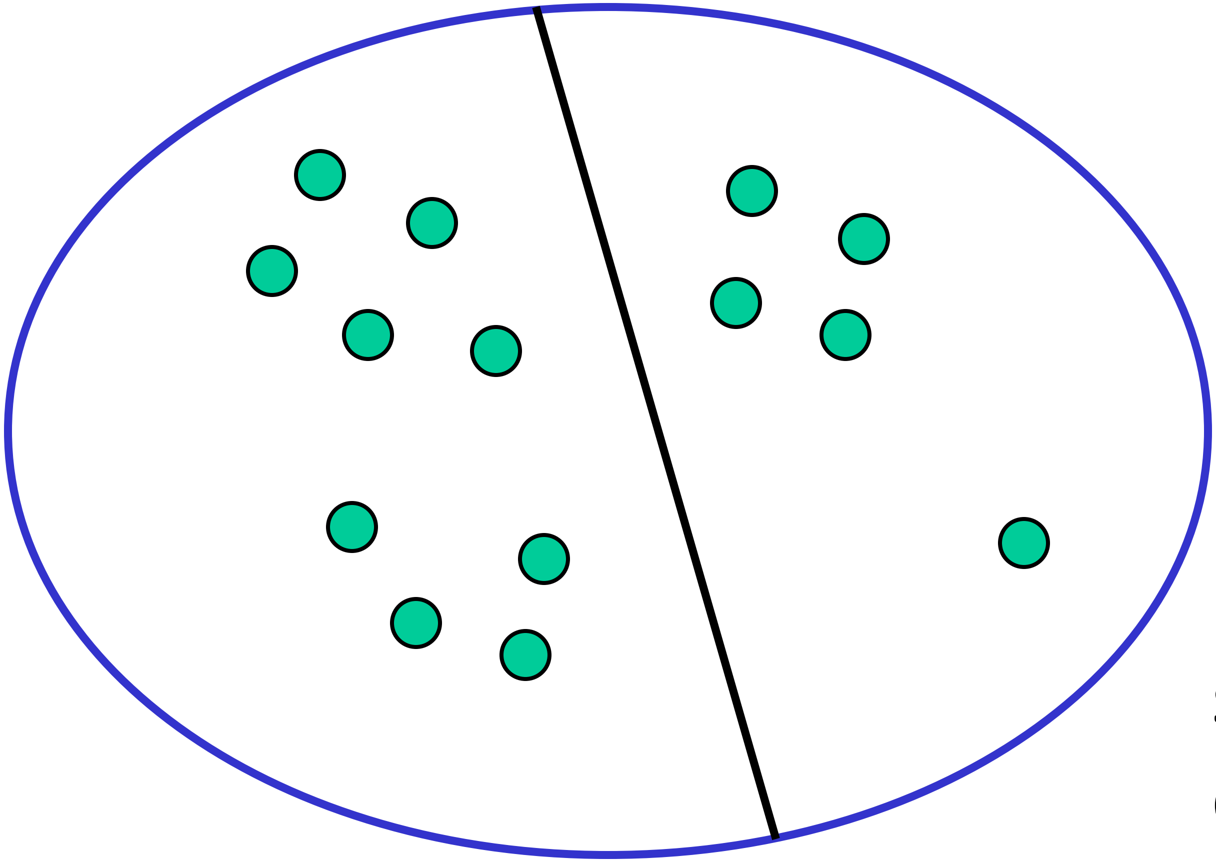


Divisive clustering



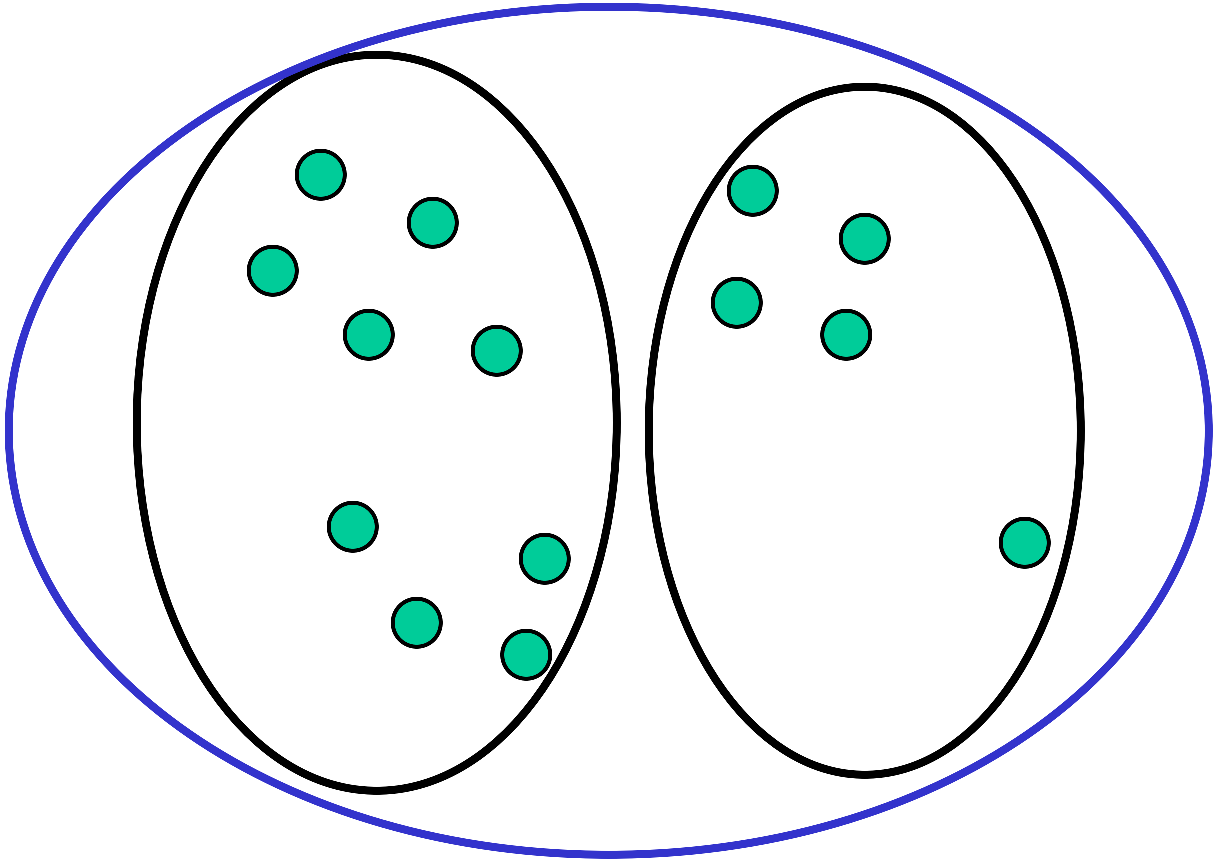
start with one cluster

Divisive clustering

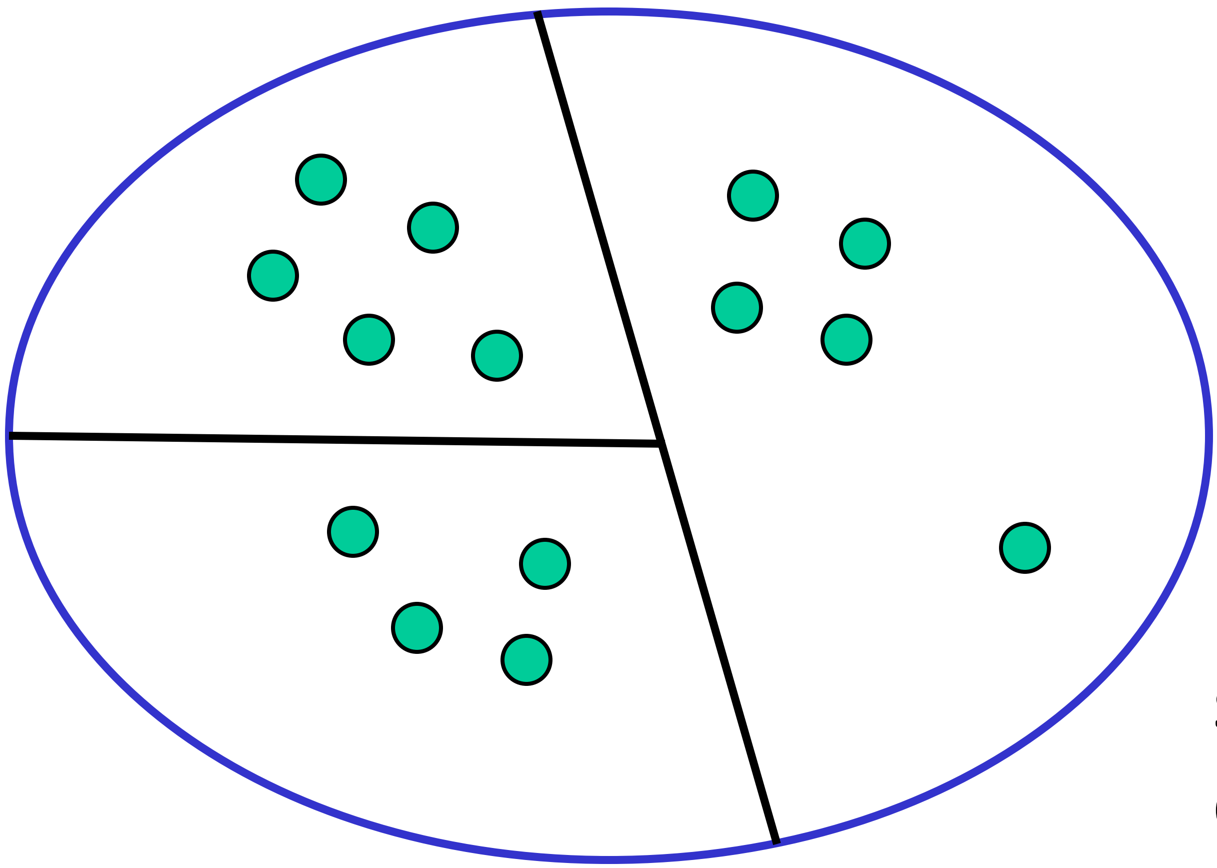


split using flat clustering, e.g., Kmeans

Divisive clustering



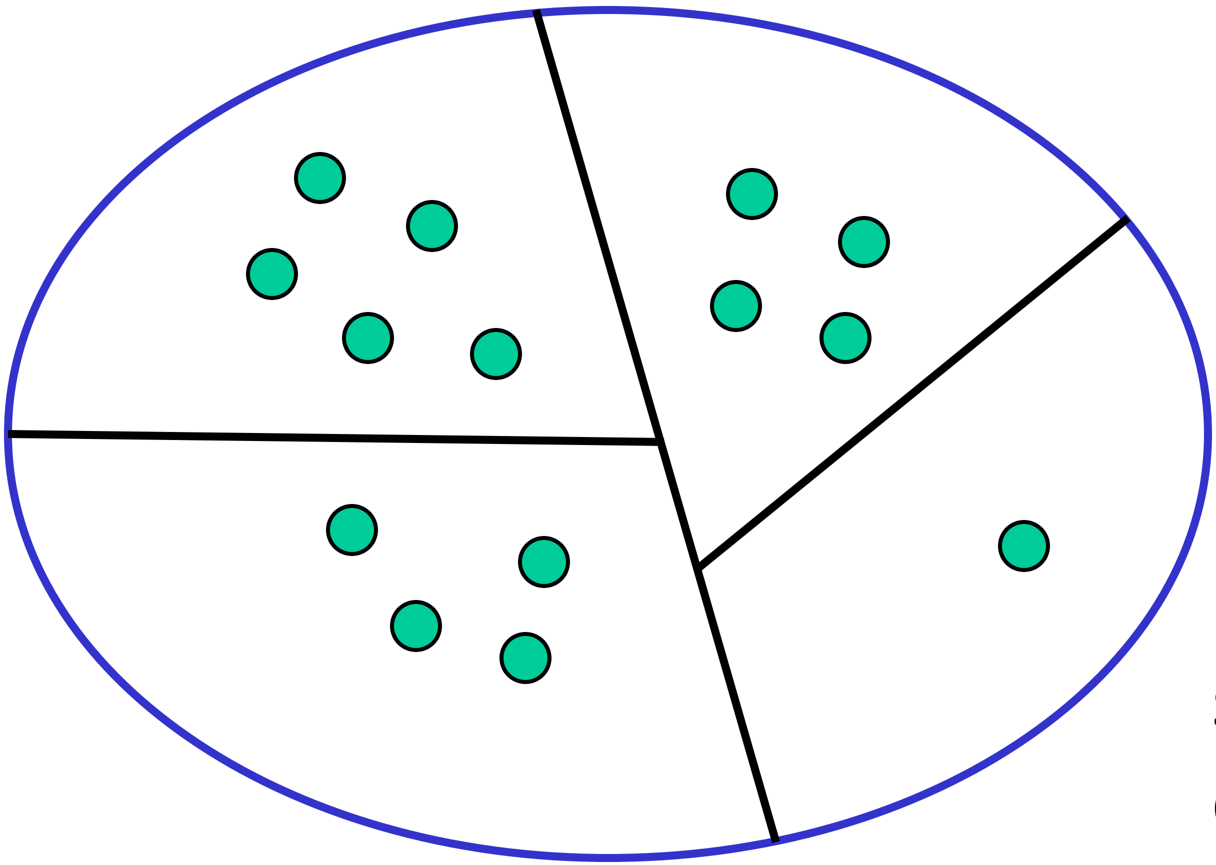
Divisive clustering



split using flat clustering, e.g., Kmeans

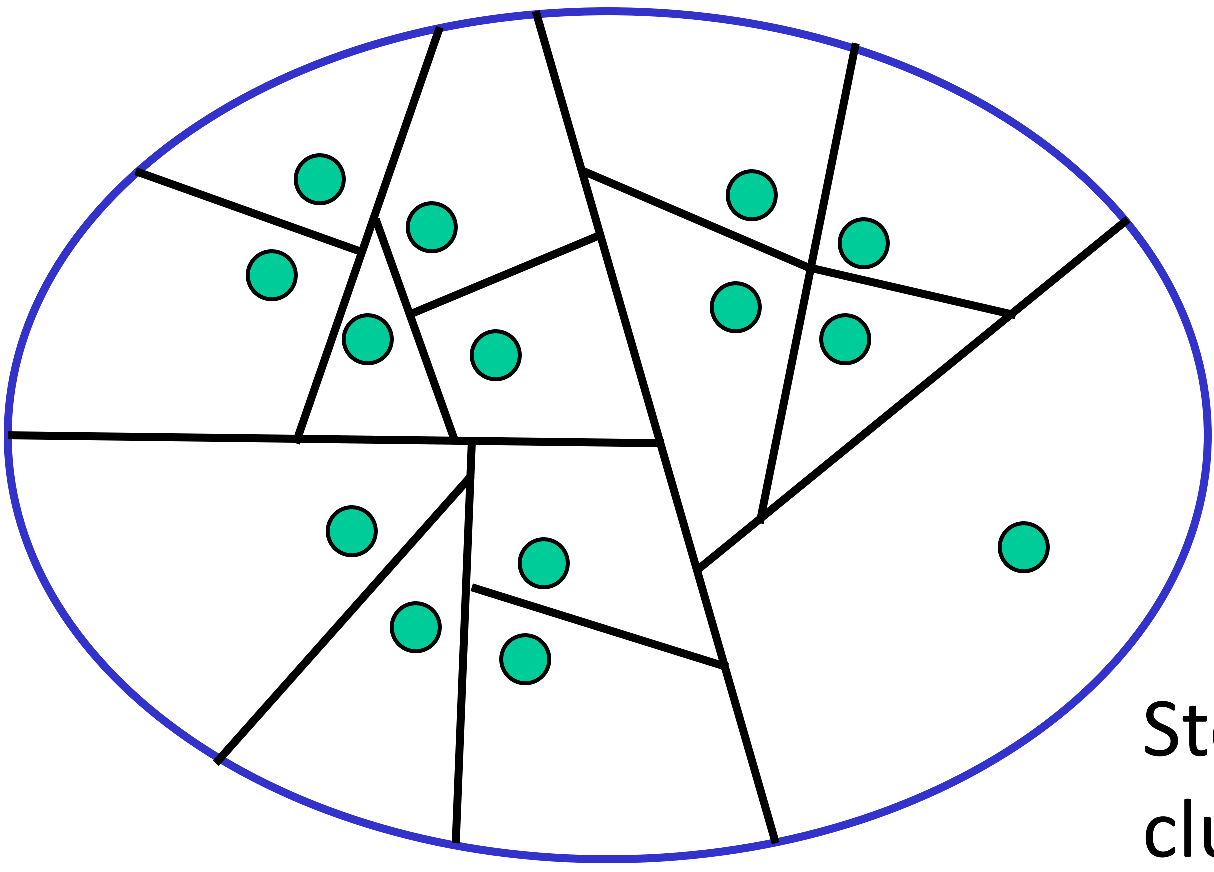
Divisive clustering

split using flat clustering



split using flat clustering, e.g., Kmeans

Divisive clustering



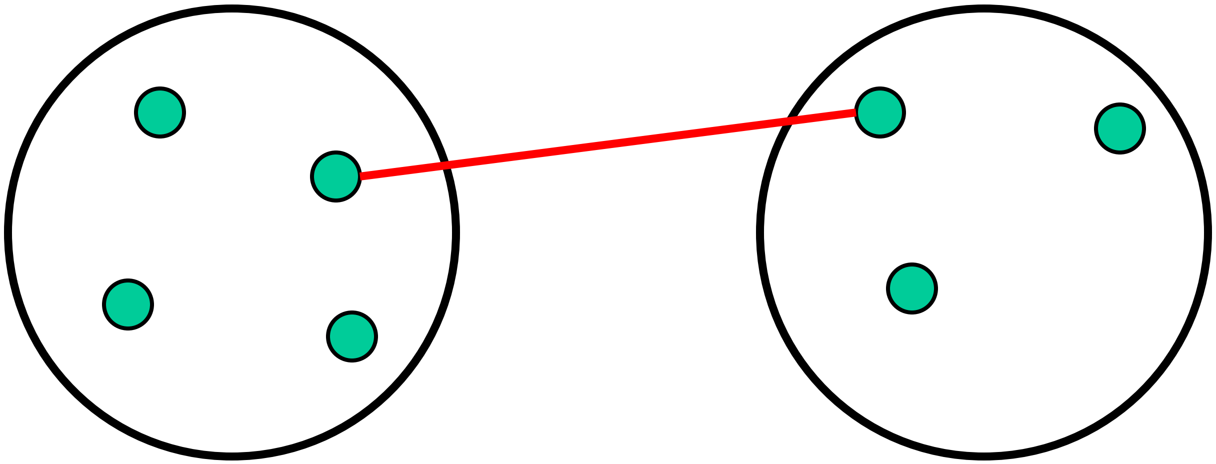
Stop when clusters reach some constraint

Hierarchical Agglomerative Clustering

- Let **C** be a set of clusters
- Initialize **C** to all points/docs as separate clusters
- While **C** contains more than one cluster
  - find c_1 and c_2 in **C** that are **closest together**
  - remove c_1 and c_2 from **C**
  - merge c_1 and c_2 and add resulting cluster to **C**
- Merging history forms a binary tree or hierarchy
- **Q: How to measure distance between clusters?**

Distance between clusters

Single-link: Similarity of the *most* similar (single-link)



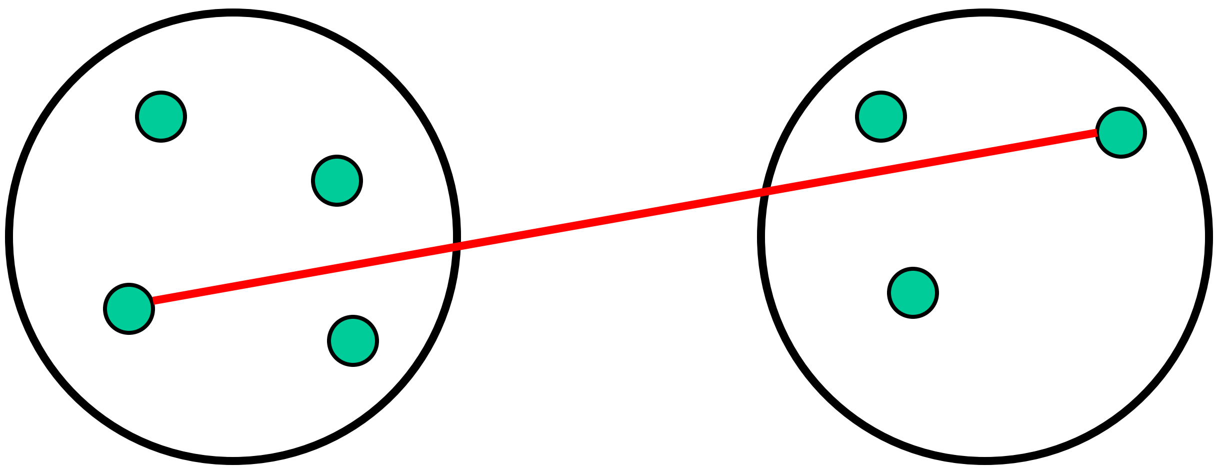
$$\max_{l \in L, r \in R} sim(l,r)$$

Weka: linkType=SINGLE

Distance between clusters

Complete-link: Similarity of the "furthest" points, the *least* similar

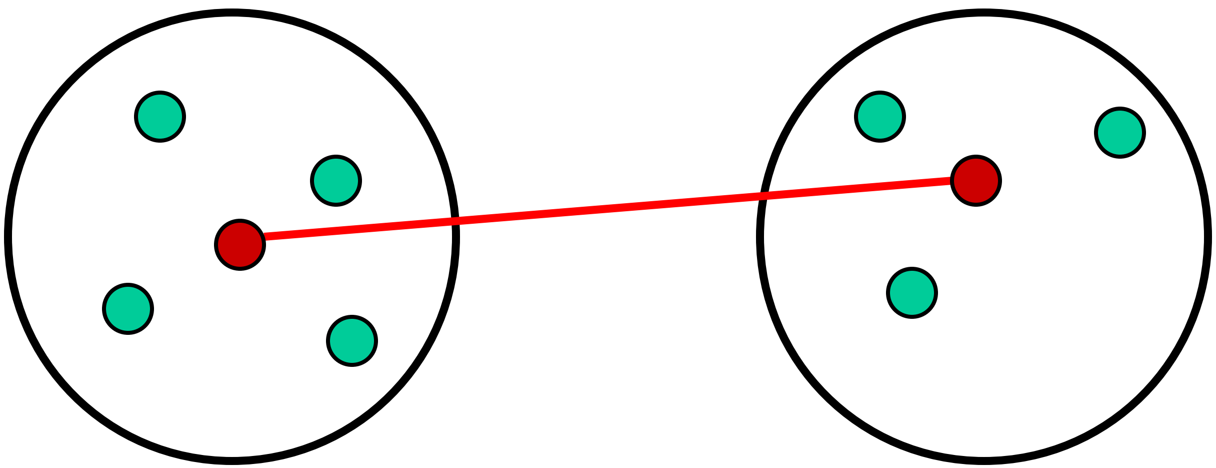
$$\min_{l \in L, r \in R} sim(l,r)$$



Weka: linkType=COMPLETE

Distance between clusters

Centroid: Clusters whose centroids (centers of gravity) are the most similar

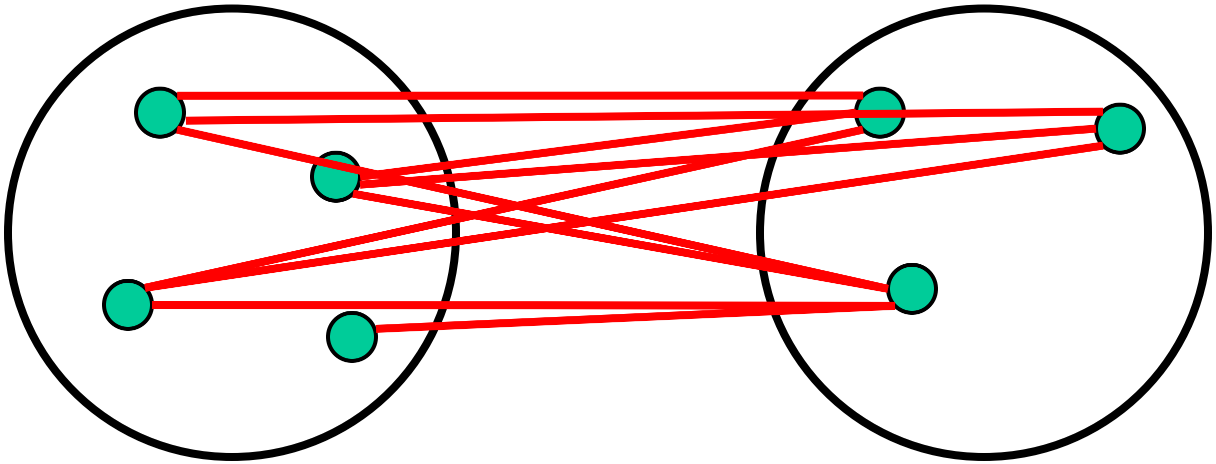


$$\|\mu(L)-\mu(R)\|^2$$

Weka: linkType=CENTROID

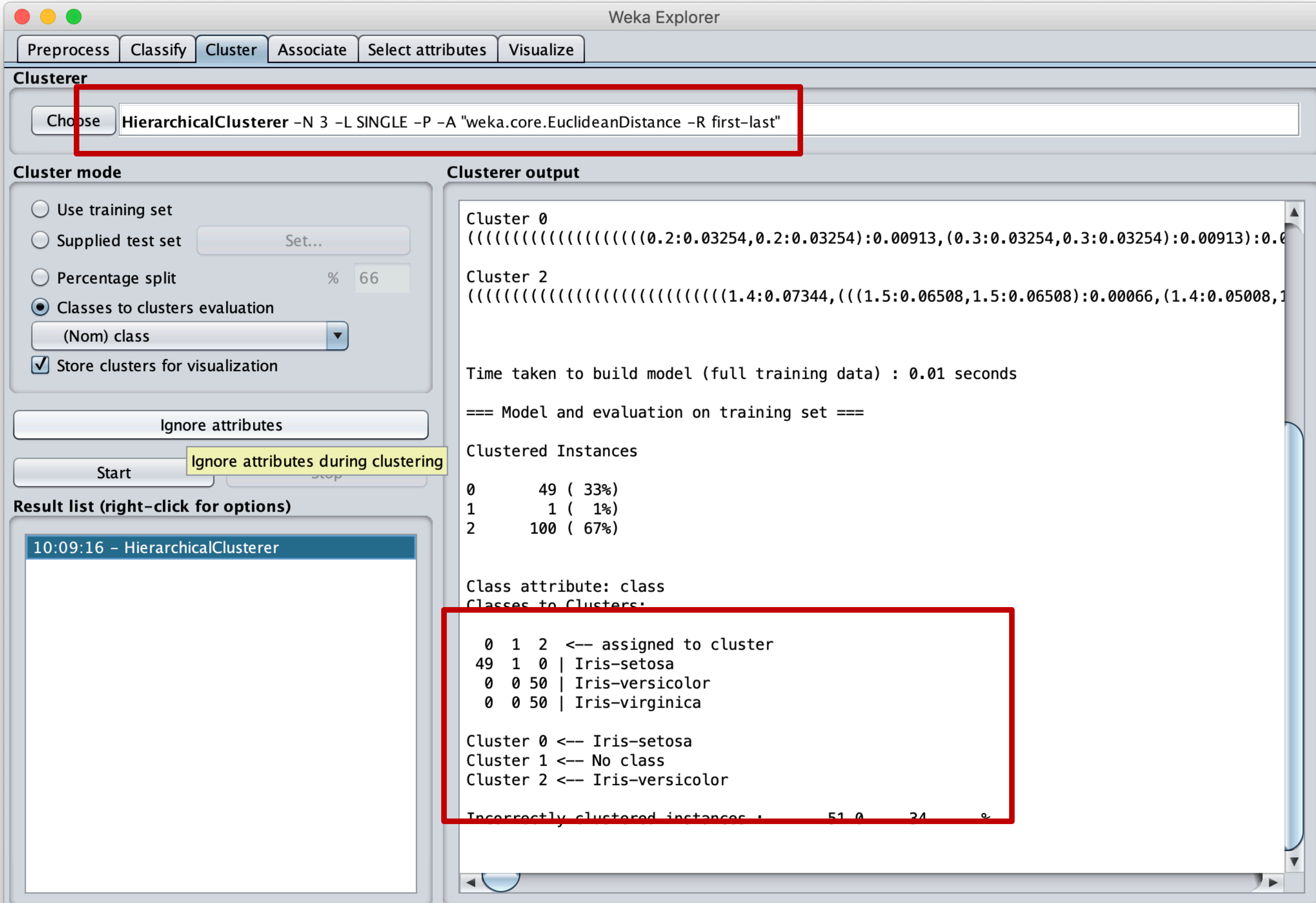
Distance between clusters

Average-link: Average similarity between all pairs of elements

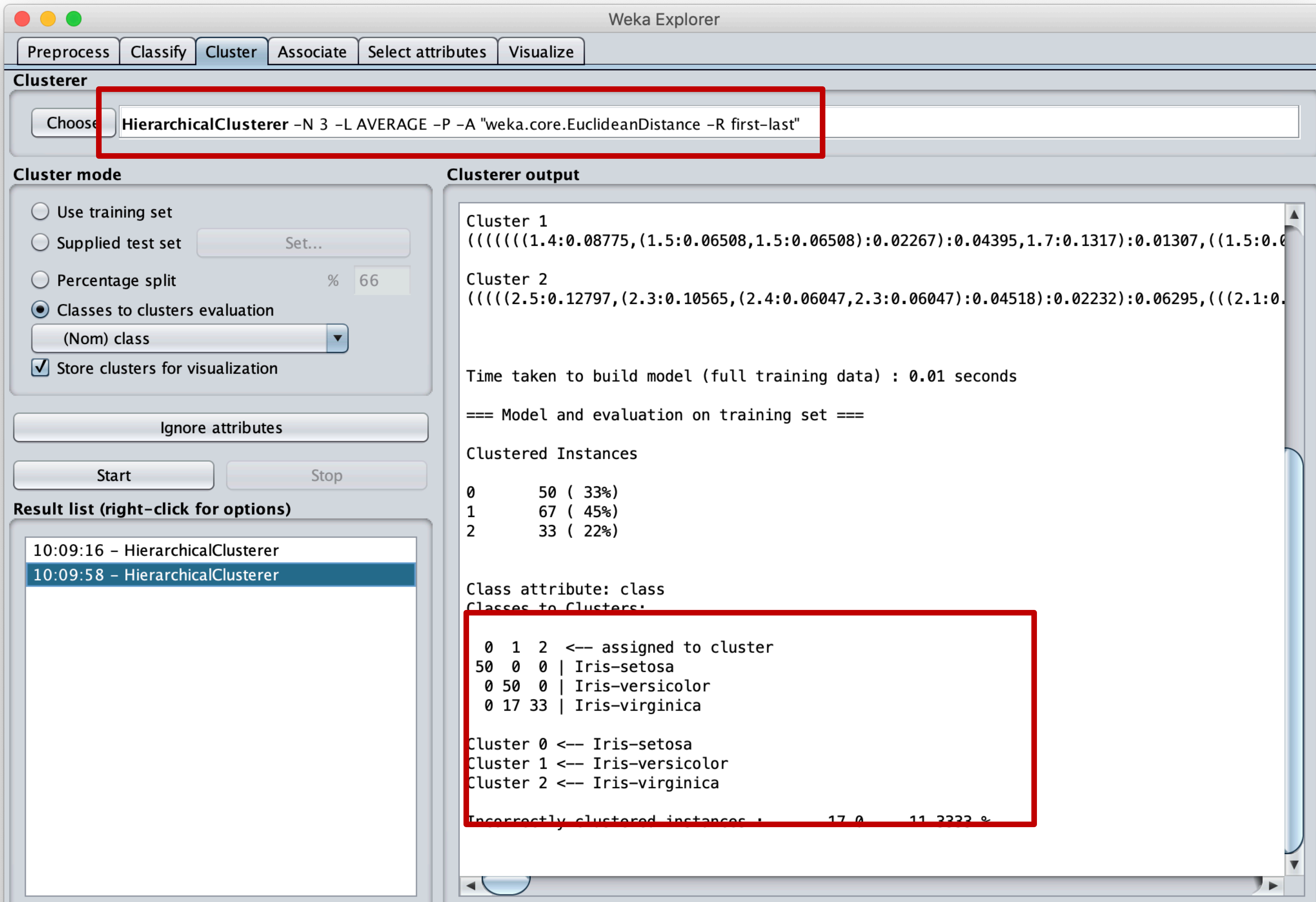


$$\frac{1}{|L| \cdot |R|} \sum_{x \in L, y \in R} \|x - y\|^2$$

Weka: linkType=AVERAGE



Defaut **SINGLE** cluster distance gives poor results here



Using **AVERAGE** cluster distance measure improves results

Knowing when to stop

- General issue is knowing when to stop merging/splitting a cluster
- We may have a problem specific desired range of clusters (e.g., 3-6)
- There are some general metrics for assessing quality of a cluster
- There are also domain specific heuristics for cluster quality