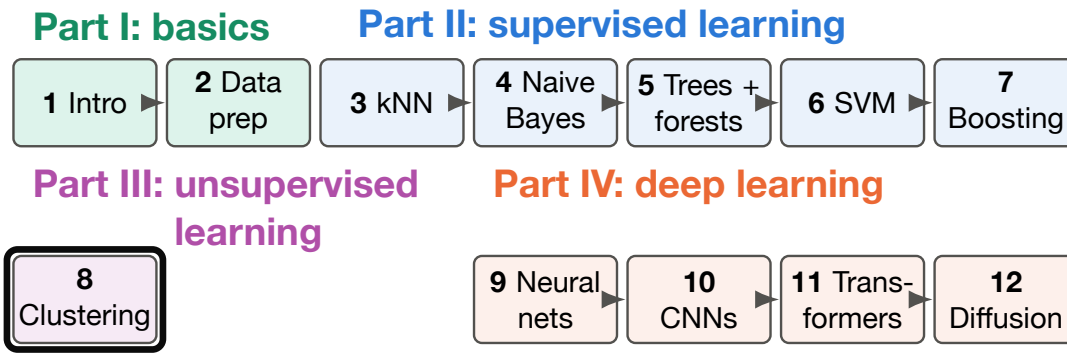
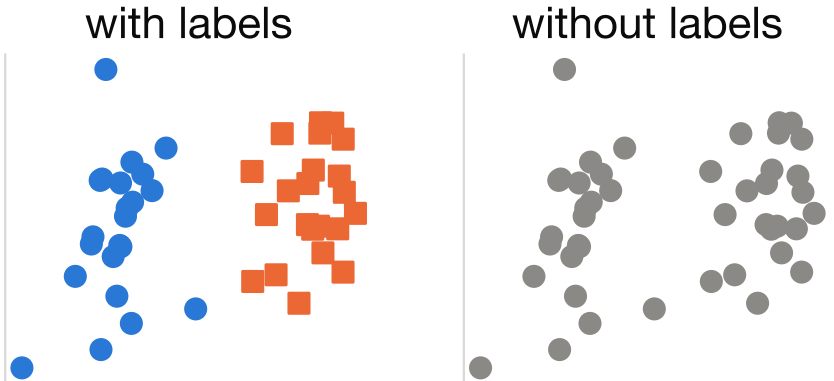
Clustering

229352 Lecture 8: clustering

Course map

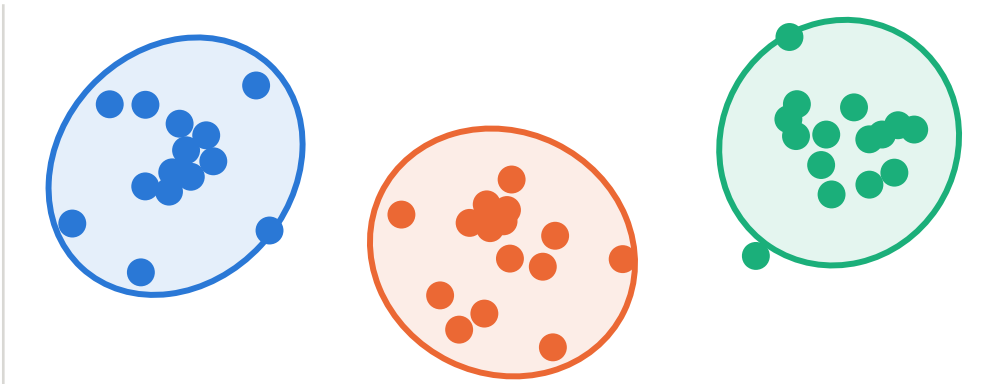

Part I: basics
Part II: supervised learning
1 Intro
2 Data prep
3 kNN
4 Naive Bayes
5 Trees + forests
6 SVM
7 Boosting
Part III: unsupervised learning
Part IV: deep learning
8 Clustering
9 Neural nets
10 CNNs
11 Trans-formers
12 Diffusion

Learning without labels



find groups using the features alone

Clustering



split the points into groups of similar points

Ways to cluster

- k-means clustering
- hierarchical clustering
- Gaussian mixture models
- density-based clustering (DBSCAN)
- spectral clustering

Today we study the first two in detail and compare k-means, single linkage, DBSCAN and Gaussian mixtures.

k-means: the goal

Data $x_1, \ldots, x_n$. Choose the number of clusters k.

Find k centers $\mu_1, \ldots, \mu_k$; each point joins the cluster of its closest center.

Make W small: the within-cluster sum of squares

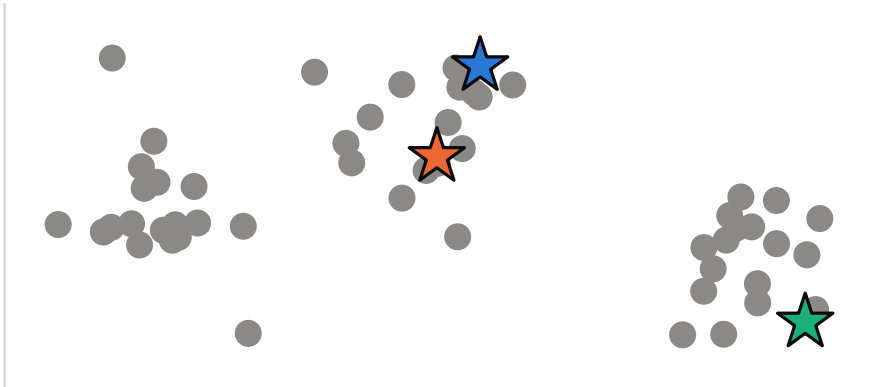
$$W = \sum_{\ell=1}^{k} \sum_{i \in C_\ell} \|x_i - \mu_\ell\|^2$$

C_ℓ: the points in cluster ℓ.

The k-means algorithm

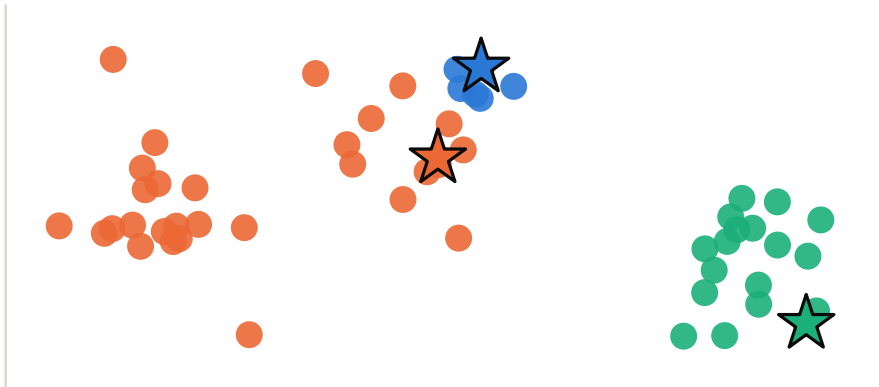
1. Pick k data points at random as the first centers.
2. **Assign**: each point joins its closest center (in a tie, the point stays in its current cluster).
3. **Move**: each center moves to the average of its points.
4. Repeat steps 2 and 3 until no point changes cluster.

Start: $k = 3$ random centers



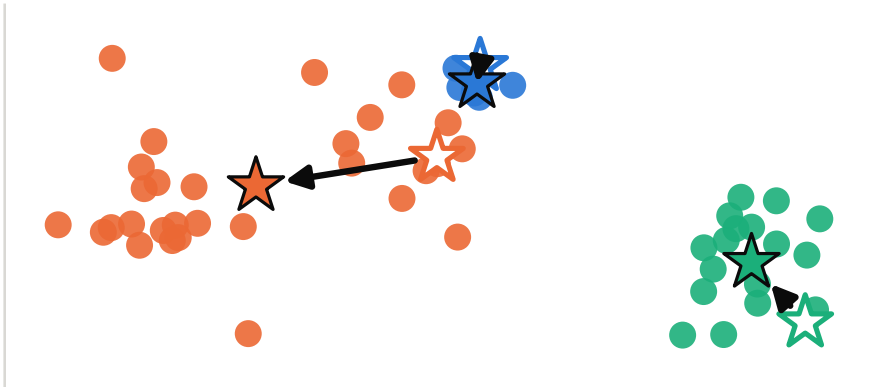
three data points, chosen at random

Assign each point to its closest center



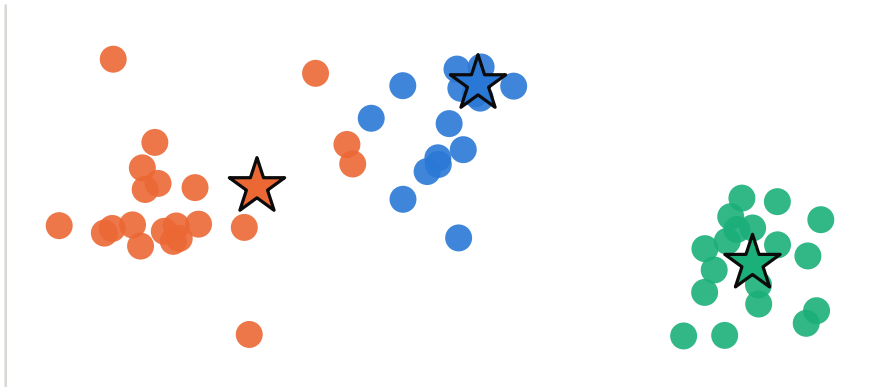
colors: the cluster of each point

Move each center to the average



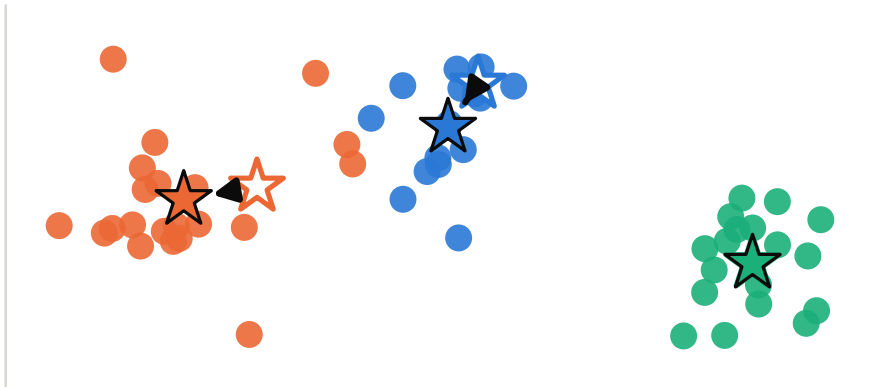
white stars: old centers; arrows: moves

Assign again



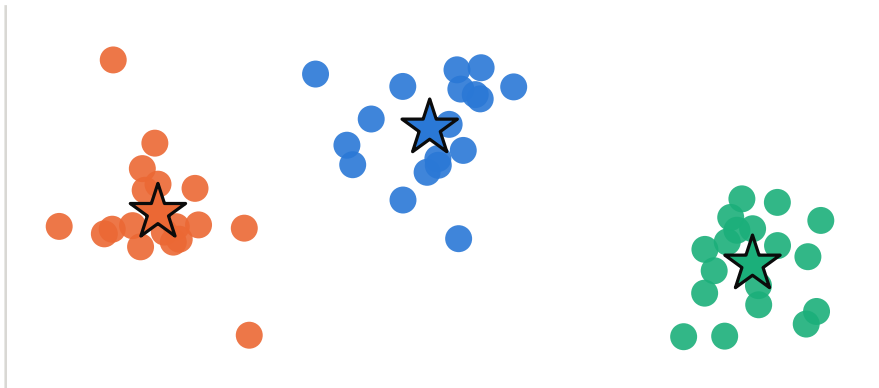
9 points change cluster

Move again



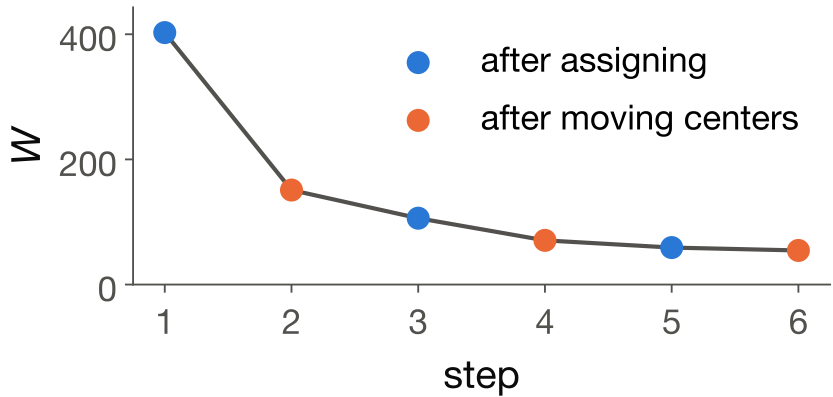
each center moves to the average of its new points

No point changes cluster: done



in the assign steps, 54 (first assignment), 9, 3 and then 0 points change cluster

W at each step



W goes down at every step in which a point or a center changes

Why k-means stops

- **Assign**: a point changes cluster only to a strictly closer center, so W strictly goes down.
- **Move**: the average of a cluster's points is the μ with the smallest $\sum_{i \in C_\ell} \|x_i - \mu\|^2$, so W does not go up.
- So the algorithm never repeats a **partition** into clusters.
- There are finitely many partitions of n points into k clusters, so the algorithm stops.
- The algorithm can stop at a partition that no step changes, even though another partition has a smaller W. Such a partition is a **local minimum**.

Summary: k-means

- Choose k. Look for k centers that make the within-cluster sum of squares W small.
- Repeat: assign each point to its closest center; move each center to the average of its points.
- W strictly goes down whenever a point changes cluster and there are finitely many partitions, so k-means stops; the result can be a local minimum.

Checkpoint 1

k-means with $k = n$: one cluster per point

$$W = ?$$

Checkpoint 1

k-means with $k = n$: one cluster per point

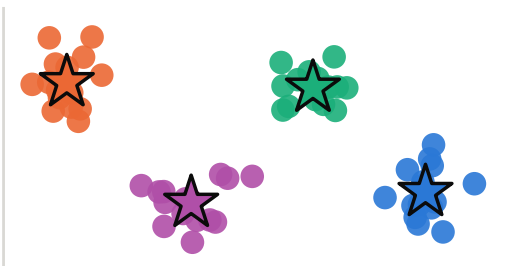
$$W = ?$$

0: every point is its own center

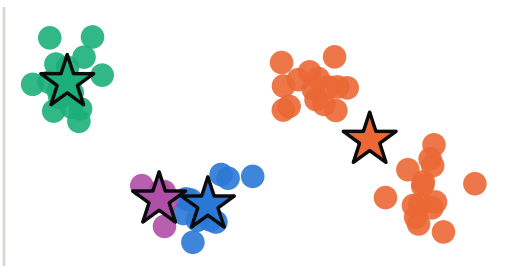
so choosing the k with the smallest W always gives $k = n$

The start matters

start A: $W = 45$

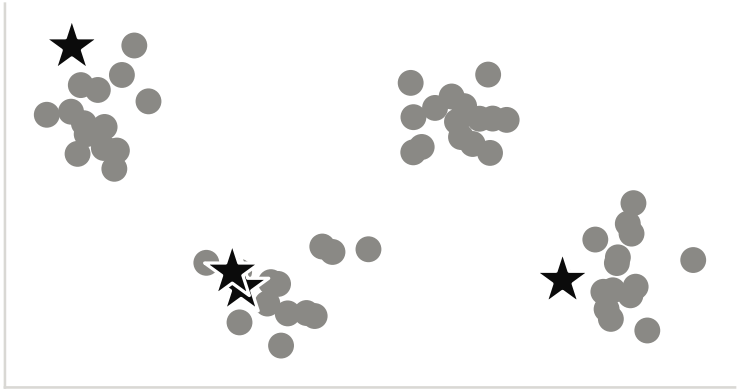


start B: $W = 176$



Two starts give two results (same data, $k = 4$)

Random start



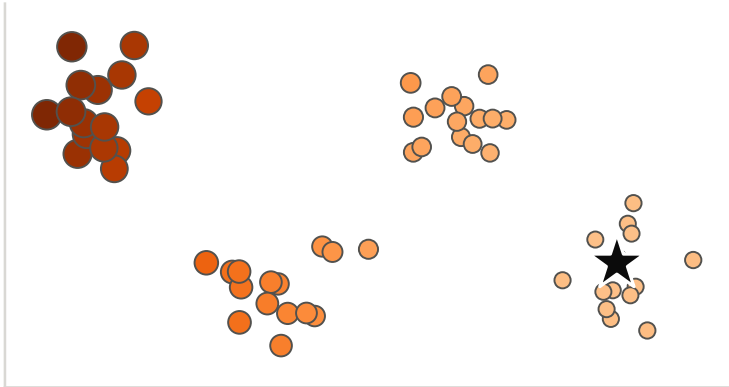
two starting centers in the same group

k-means++: spread out the start

1. The first center is a data point chosen at random.
2. Each next center is a data point x drawn with probability $D(x)^2 / \sum_i D(x_i)^2$, where $D(x)$ is the distance from x to the closest center chosen so far.
3. So points with large $D(x)$ are likely to be drawn.
4. Stop at k centers, then run k-means from them.

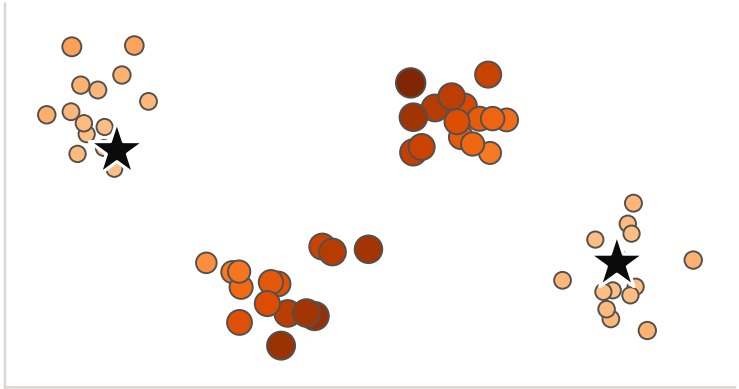
Arthur and Vassilvitskii, *k-means++: the advantages of careful seeding*, SODA 2007

k-means++: the first center



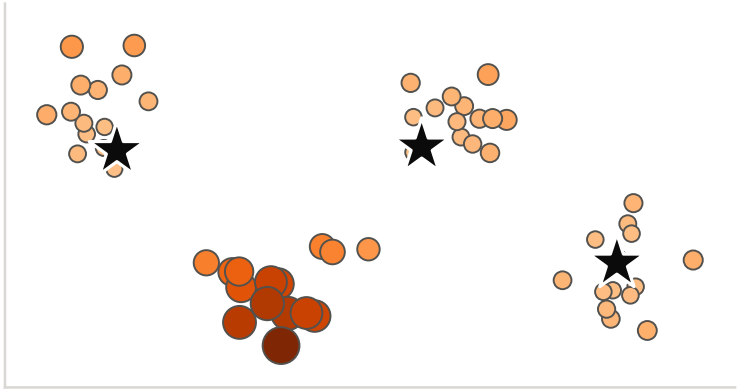
darker and larger: more likely to be the next center

k-means++: the second center



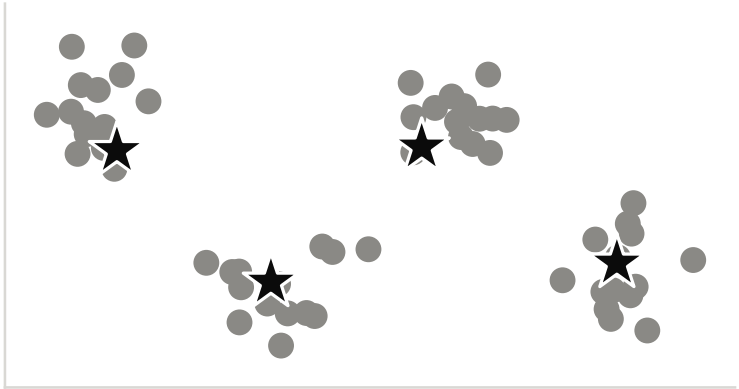
points near a center are unlikely to be drawn

k-means++: the third center



the next center is likely in the group without one

k-means++: four centers



one center in each group

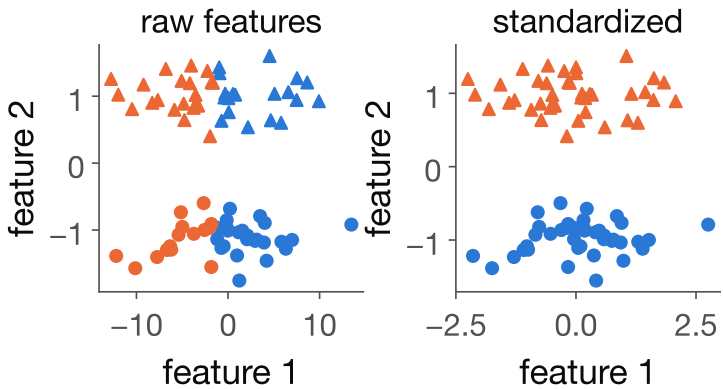
Random start versus k-means++

Same data as before; 1000 starts of each kind.

	random	**k-means++**
one center in each group	9.7%	77.7%
ends at the best W	60.3%	92.5%

In practice: run several starts; keep the smallest W.

Standardize the features first



shape: true group; color: k-means cluster

Standardize the features

x_{ij}: feature j of point i; feature j has mean m_j and standard deviation s_j.

$$x_{ij} \longmapsto \frac{x_{ij} - m_j}{s_j}$$

In these data, $s_1 = 5.3$ and $s_2 = 1.1$, so feature 1 dominates the distances.

Points whose 2-means cluster agrees with their true group: 57% (raw), 100% (standardized).

Standardize when the features are in different units.

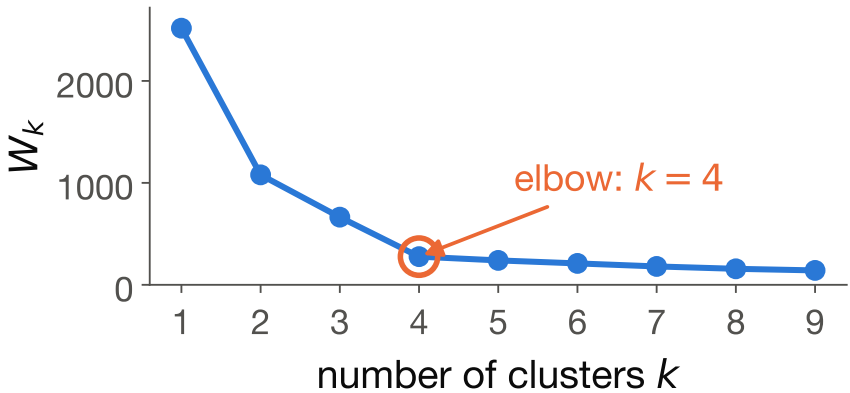
Choosing k

W_k: the smallest W with k clusters (in practice: the best of several starts).

W_k never goes up as k grows, and $W_n = 0$ (Checkpoint 1).

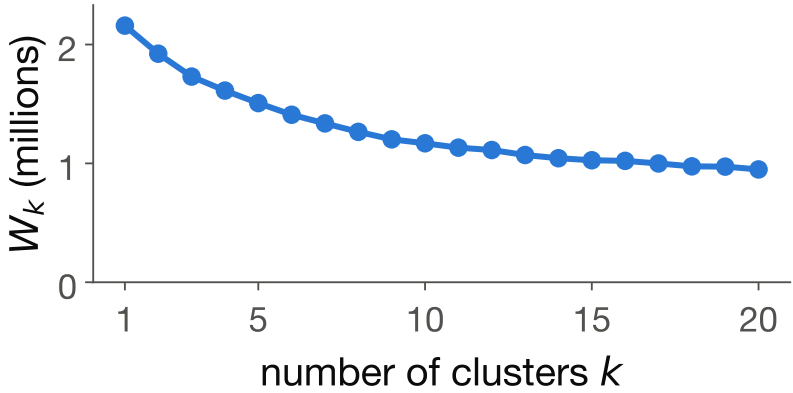
So look for an **elbow**: the k after which more clusters lower W_k only a little.

W_k as k increases



W_k decreases quickly up to $k = 4$, then slowly

Real data: no clear elbow

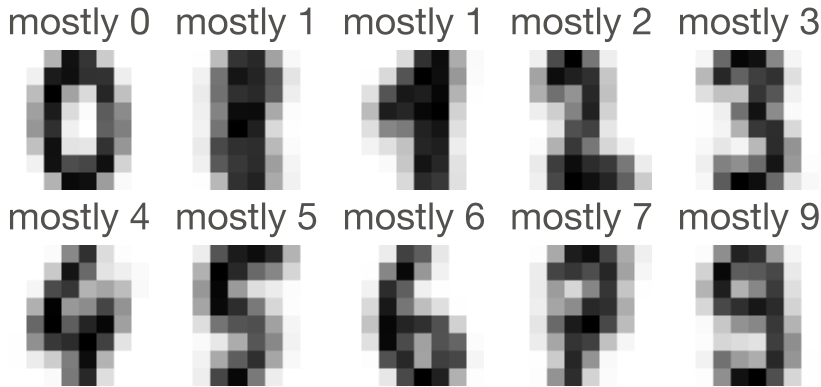


For the 1797 digit images, W_k decreases slowly

Summary: k-means in practice

- The result depends on the start: use k-means++, and run several starts.
- A feature with a large scale dominates the distances: standardize features that are in different units.
- Choose k at the elbow of W_k; real data may have no clear elbow.

Digits without labels



the 10 centers of k-means with $k = 10$ (10-means) on 1797 digit images (8×8 pixels)

What 10-means found

- Each image is a point with 64 pixel values; k-means groups similar images.
- 79% of the images are in a cluster whose most common digit is the image's own digit.
- Two clusters are mostly 1s, and no cluster is mostly 8s.
- Similar pixel values need not mean the same digit.

Training on fewer points

- An SVM with an RBF kernel uses a kernel matrix: $n \times n$ entries for n training images.
- MNIST: $n = 60{,}000$, so 3.6 billion entries.
- Idea: within each digit, run k-means with $k = 10$; train the SVM on the 100 centers, each labeled with its digit (this uses the labels).
- Digits data: 1347 training images $\rightarrow$ 100 centers; 450 test images.

Training on 100 centers

5 of the 136 training images of 5

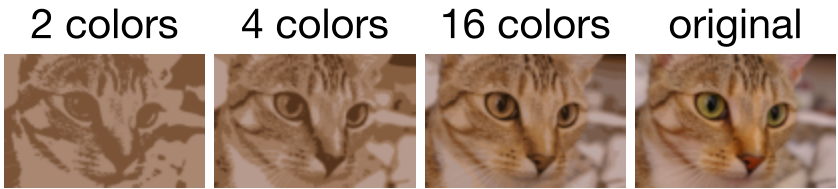


5 of the 10 centers that replace all of them



trained on	kernel entries	test accuracy
1347 images	1,814,409	99.1% (446/450)
100 centers	10,000	97.8% (440/450)

Image compression

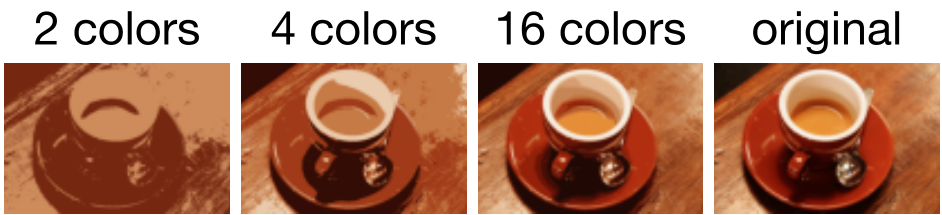


Pixel colors are points (R, G, B); the k centers form a palette.

Each pixel stores the index of its center: $\lceil \log_2 k \rceil$ bits instead of 24, plus a palette of $24k$ bits.

Photo: scikit-image sample image “chelsea”, CC0 (S. van der Walt)

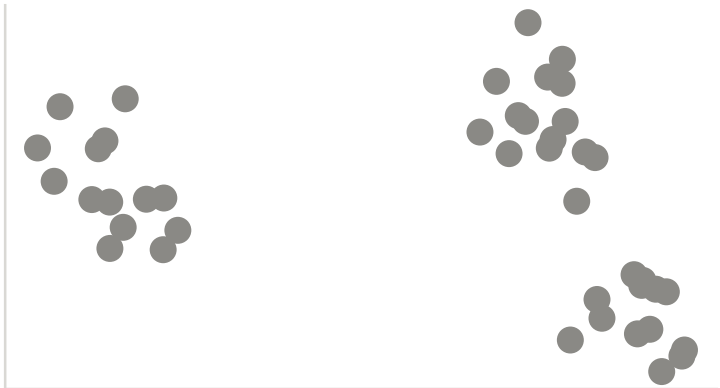
Image compression: a second photo



2 colors: 1 bit per pixel; 4 colors: 2 bits; 16 colors: 4 bits.

Photo: scikit-image sample image “coffee”, CC0 (R. Michetti)

Two groups or three?



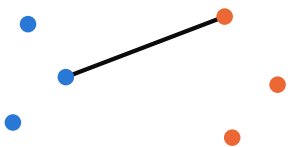
hierarchical clustering: the clusters for every k

Hierarchical clustering: the algorithm

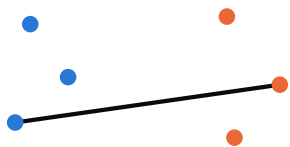
1. Start with every point as its own cluster.
2. Merge the two closest clusters; record the distance between them.
3. Repeat until one cluster is left.
4. Draw the merges as a tree: the **dendrogram**.

We still need the distance between two clusters.

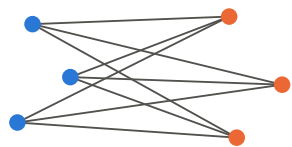
Distance between two clusters



single



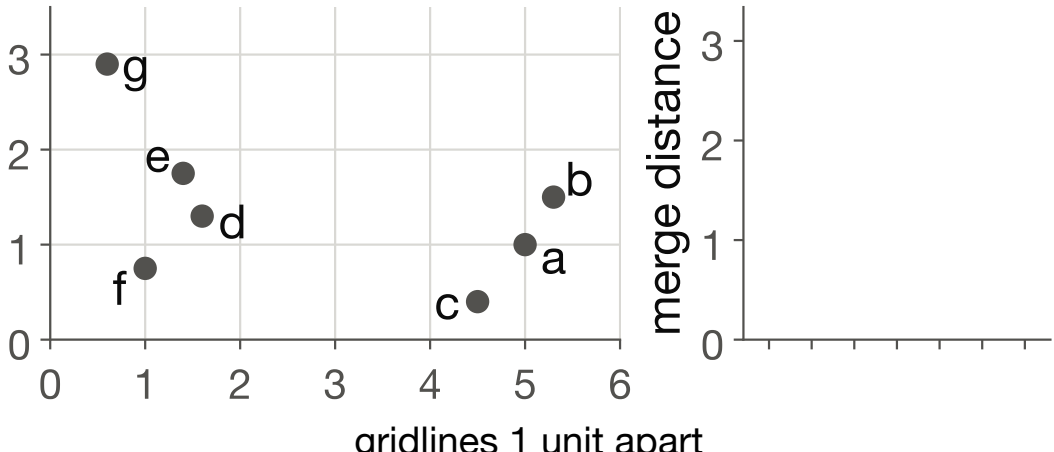
complete



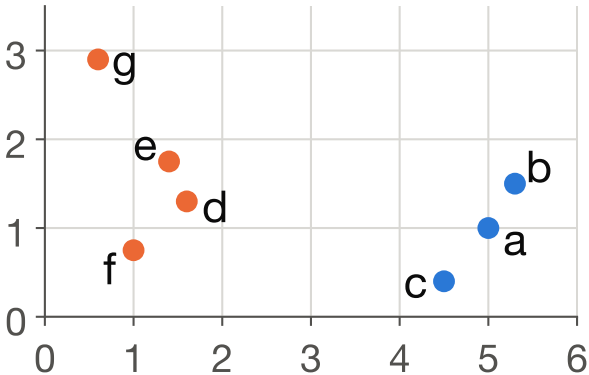
average

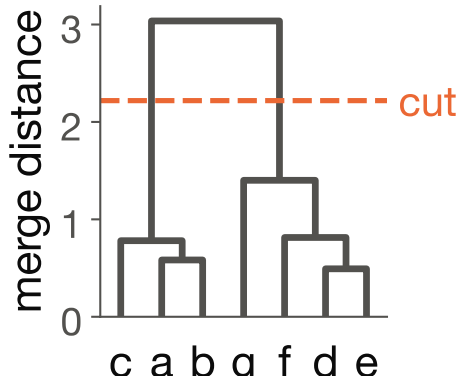
- **single**: the closest pair, one point from each cluster
- **complete**: the farthest such pair
- **average**: the average over all such pairs

Build the dendrogram (single linkage)



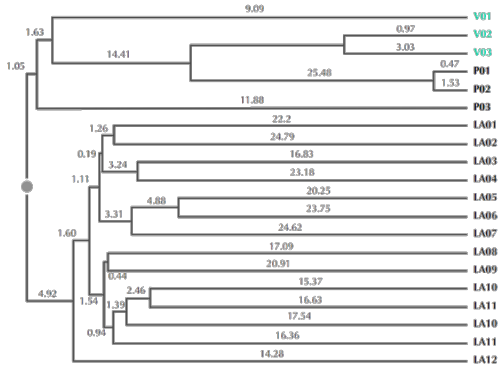
Cut the dendrogram





cut at 2.2: clusters $\{a, b, c\}$ and $\{d, e, f, g\}$

Trees of virus sequences

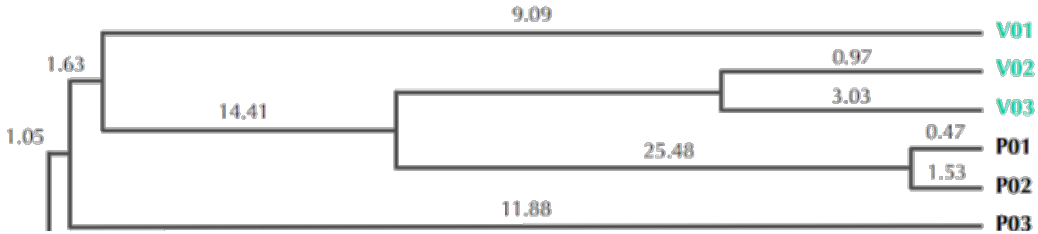


Metzker et al., *Molecular evidence of HIV-1 transmission in a criminal case*, PNAS 99 (2002), 14292–14297

HIV sequences from a victim (V), a patient whose blood was the suspected source (P), and local controls (LA).

This is a phylogenetic tree: similar sequences join near the leaves, as in a dendrogram.

Zoom: the V and P sequences



Top of the tree in Metzker et al., *Molecular evidence of HIV-1 transmission in a criminal case*, PNAS 99 (2002), 14292–14297

The V and P sequences form one branch, apart from all the controls.

Summary: hierarchical clustering

- Start from single points; repeatedly merge the two closest clusters.
- The distance between two clusters (single, complete or average linkage) changes the tree.
- The dendrogram records every merge; a horizontal cut gives the clusters for one k.

Checkpoint 2

Two well-separated groups, and a line of closely spaced points between them

What does single linkage do?

Checkpoint 2

Two well-separated groups, and a line of closely spaced points between them

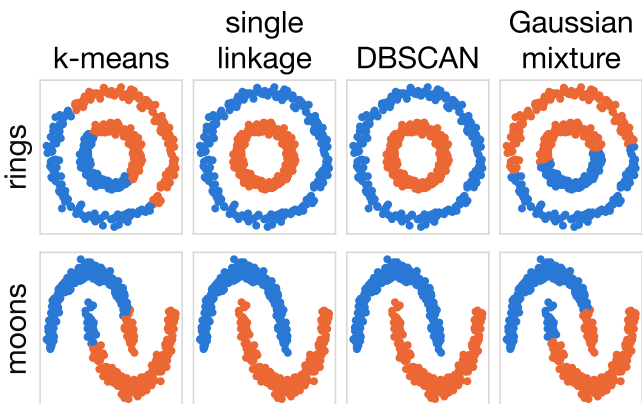
What does single linkage do?

If the gaps on the line are as small as the gaps inside the groups, single linkage joins the points along the line one after another: the two groups merge at a small distance, so no cut at one height separates them.

Two more methods

- **Gaussian mixture**: each cluster is a bell-shaped (Gaussian) cloud with its own center and spread; each point gets a probability for each cluster.
- **DBSCAN**: a cluster is a region where points lie close together; points in sparse regions are called noise.

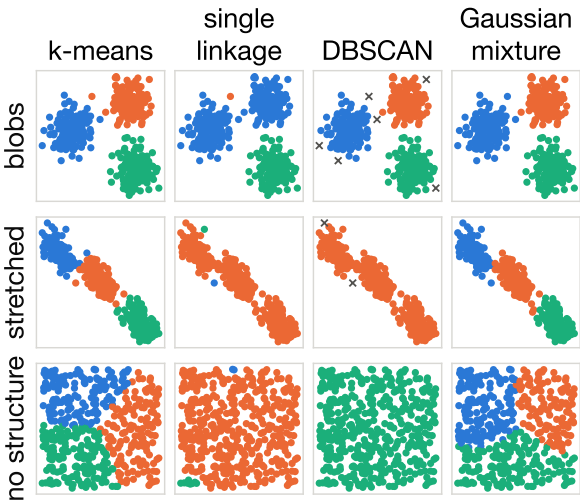
Comparing clustering methods (1)



colors: the clusters each method found.

k-means and the Gaussian mixture split each ring and each moon into the wrong groups.

Comparing clustering methods (2)



$\times$: points DBSCAN calls noise.

Single linkage merges two blobs that nearly touch.

No structure: k-means and the Gaussian mixture still return three groups; DBSCAN returns one.

What the comparison shows

- k-means and Gaussian mixtures look for round or oval groups, and always return k of them.
- Single linkage and DBSCAN follow chains of nearby points: they find the rings and the moons, but merge groups that touch.
- No method is best on every data set.

Figures after the scikit-learn toy-dataset clustering comparison

Clustering in scikit-learn

```
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans, AgglomerativeClustering
from scipy.cluster.hierarchy import linkage, dendrogram

Xs = StandardScaler().fit_transform(X)  # standardize

km = KMeans(n_clusters=4, init="k-means++", n_init=10)
km.fit(Xs)  # uses k-means++
km.labels_, km.cluster_centers_, km.inertia_  # inertia_ = W

# hierarchical clustering, single linkage
agg = AgglomerativeClustering(n_clusters=2, linkage="single")
agg.fit(Xs)
dendrogram(linkage(Xs, method="single"))  # draw the tree
```