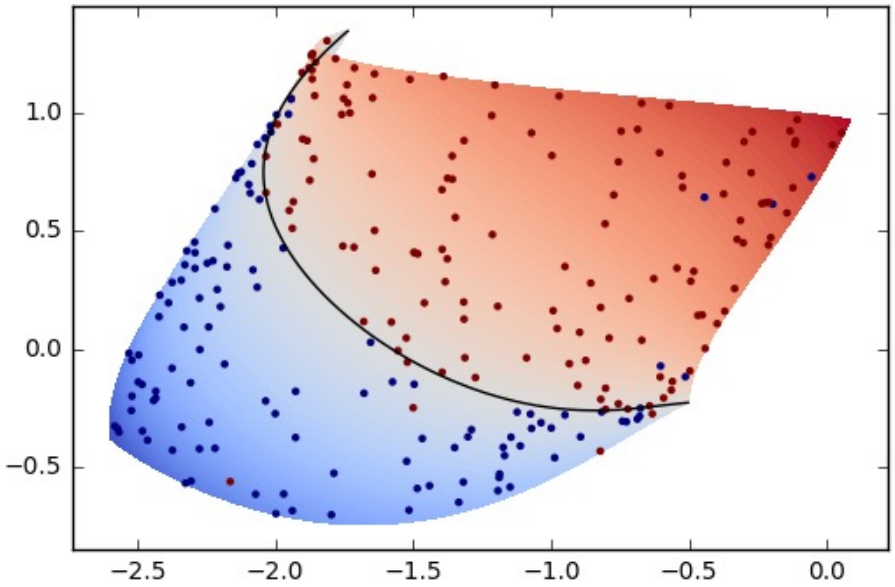
Lecture 23: Hierarchical Clustering & Dimensionality Reduction



Gavin Kerrigan

Spring 2023

Some slides adapted from Padhraic Smyth, Alex Ihler

Announcements

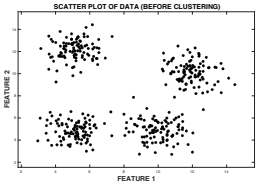
- HW4 due Friday (5/26)
  - Implementing & experimenting with decision trees
- Discussion section tomorrow:
  - Live-coding example of ensemble methods
  - Come with questions
- No class Monday (5/29) – Memorial day

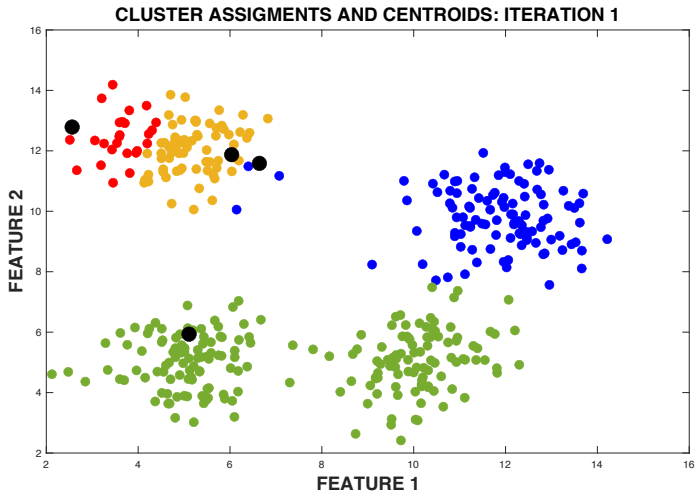
kMeans++

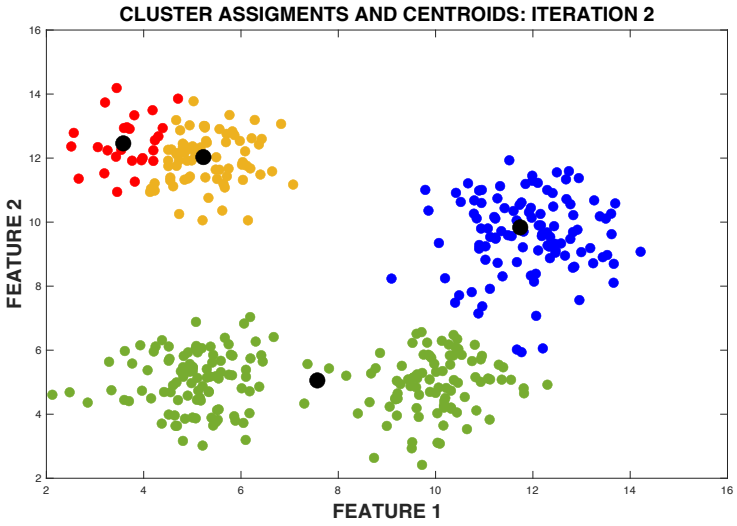
Hierarchical Clustering

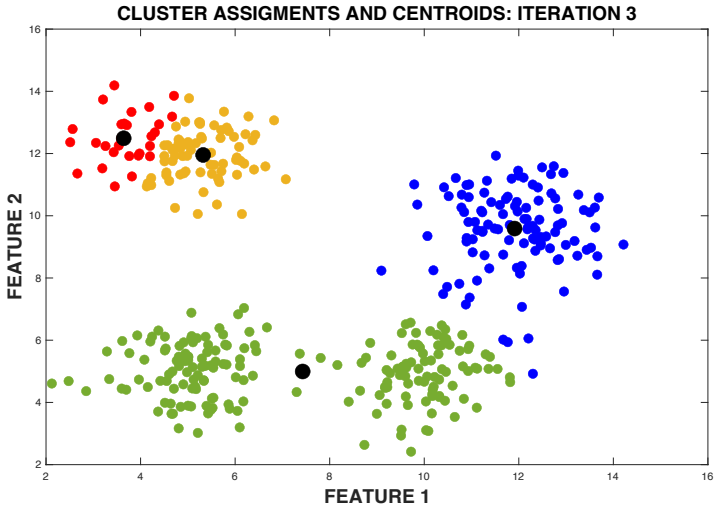
Dimensionality Reduction

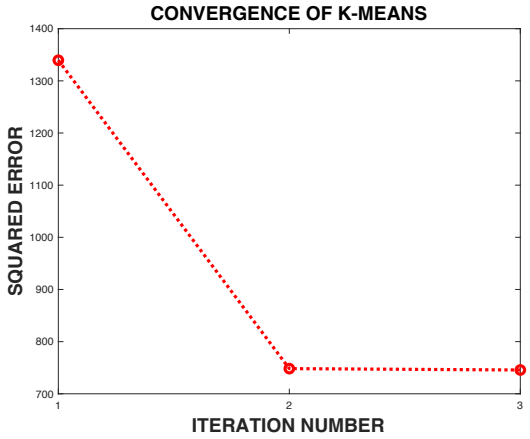
Local Minimum











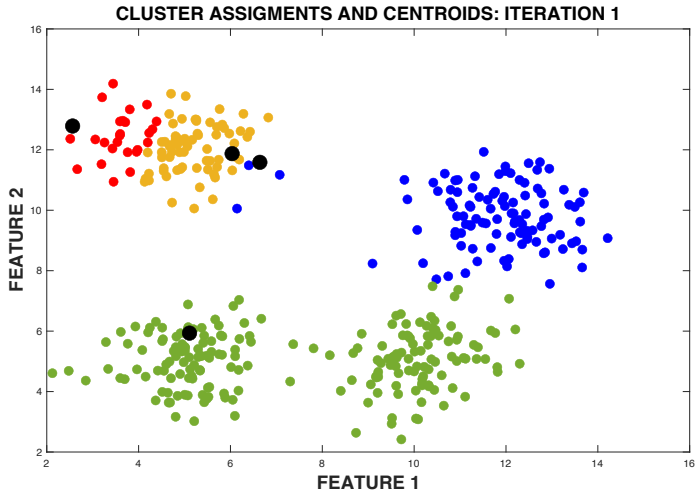
kMeans++

One strategy for avoiding local minima is to initialize clusters in a smart way

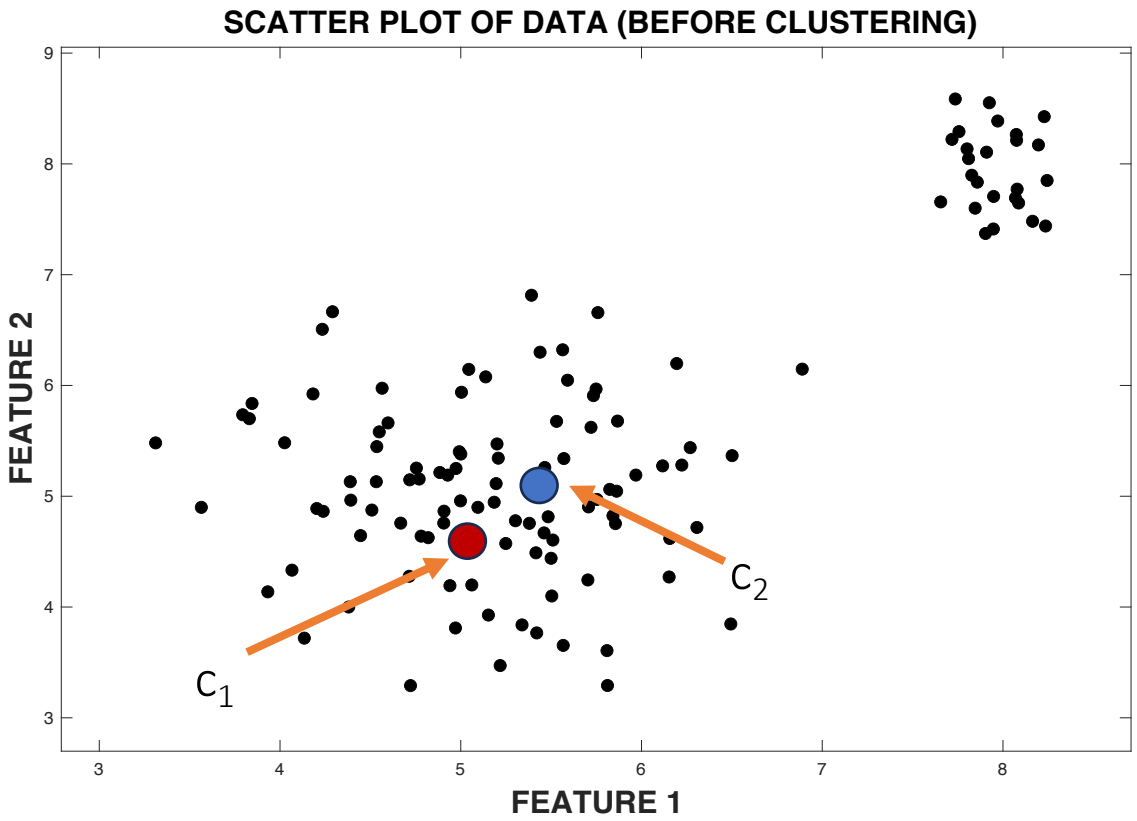
Recall: standard approach is to initialize clusters with random datapoints

Problem: initial clusters can be bad guesses

- Want centroids that are far apart

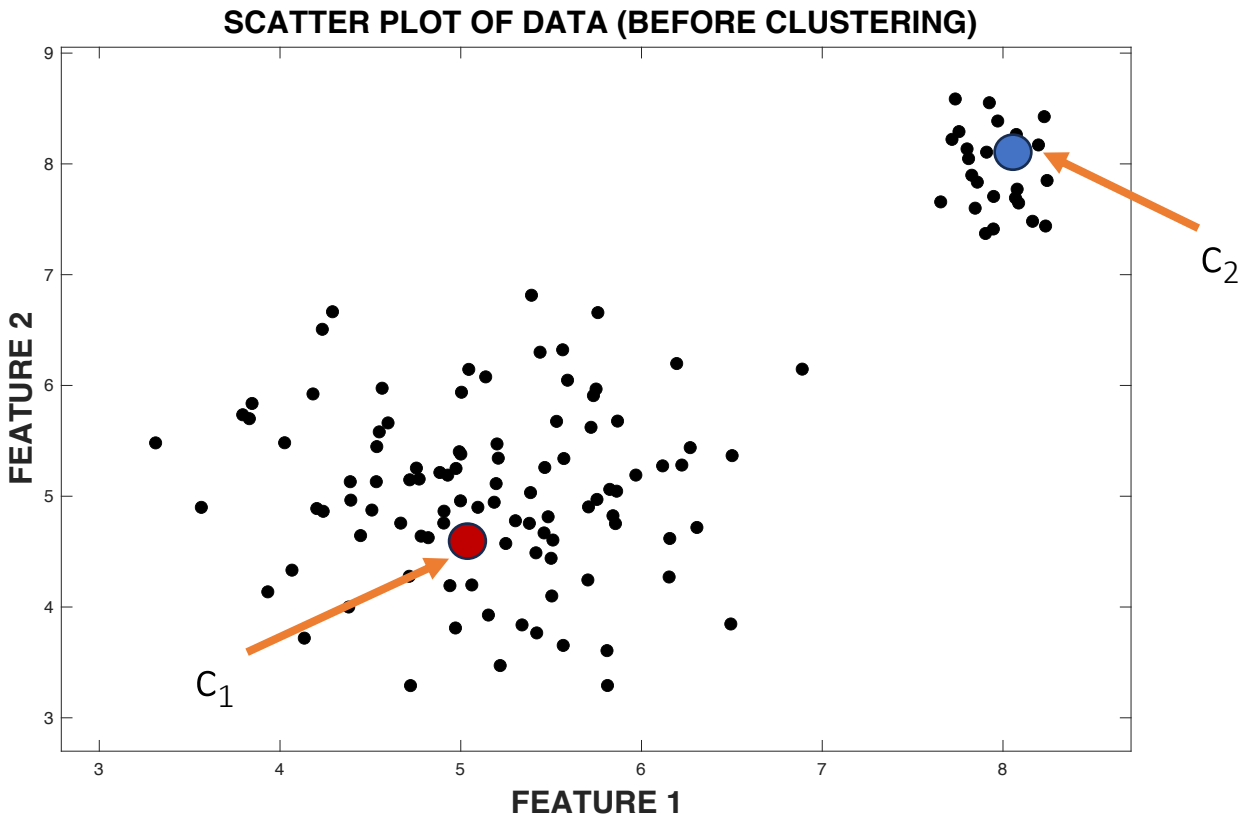


Example: K-Means Clustering, 2 Clusters



Unlucky initialization: centroids are nearby

Example: K-Means Clustering, 2 Clusters



Lucky initialization: centroids are far

kMeans++

kMeans++ is an alternative initialization strategy

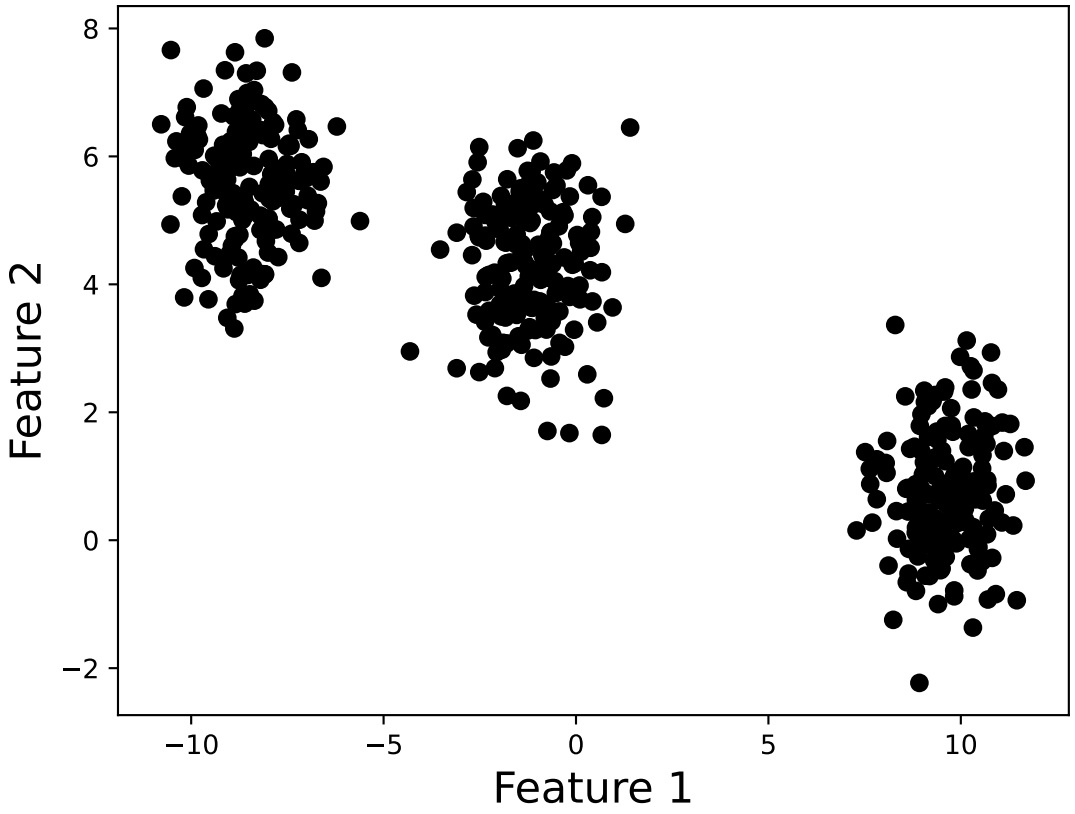
To initialize K clusters:

1. Pick the first centroid $\mathbf{c}_1$ randomly from the data
2. for k = 2, 3, . . . , K:
    1. For each datapoint $\mathbf{x}_i$ not yet chosen as a centroid:
        1. Compute d_i, the squared Euclidean distance from $\mathbf{x}_i$ to the nearest centroid
    2. Select $\mathbf{c}_k$ randomly from the data, where the probability of choosing $\mathbf{x}_i$ is proportional to d_i

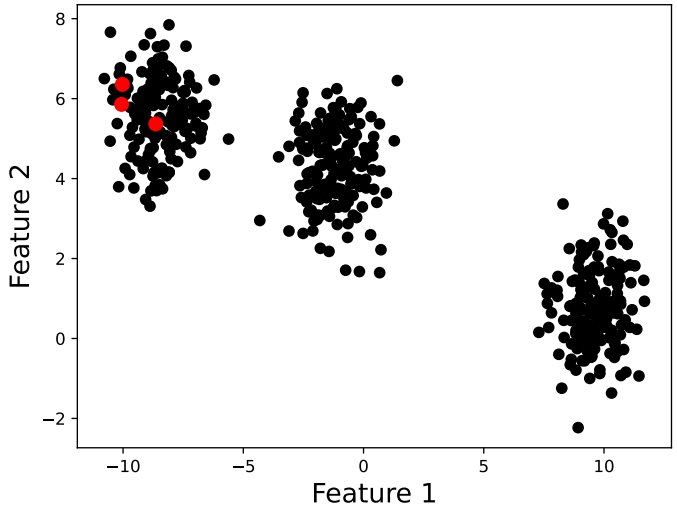
After initialization, proceed with kMeans as usual

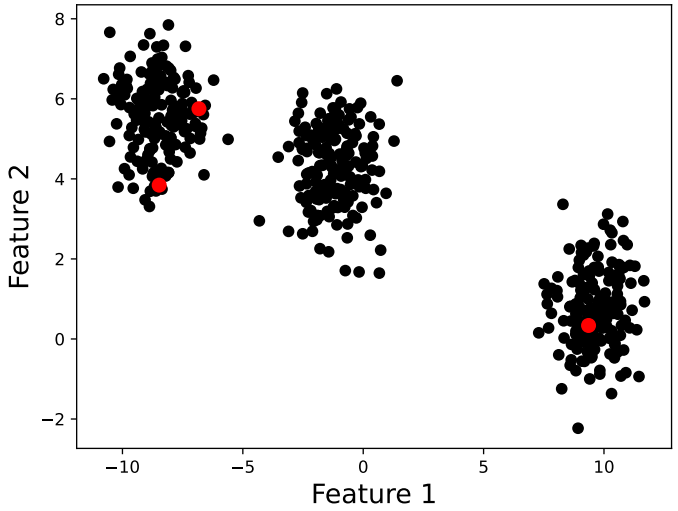
- Time complexity? O(d n K) – but only needs to be done once at initialization

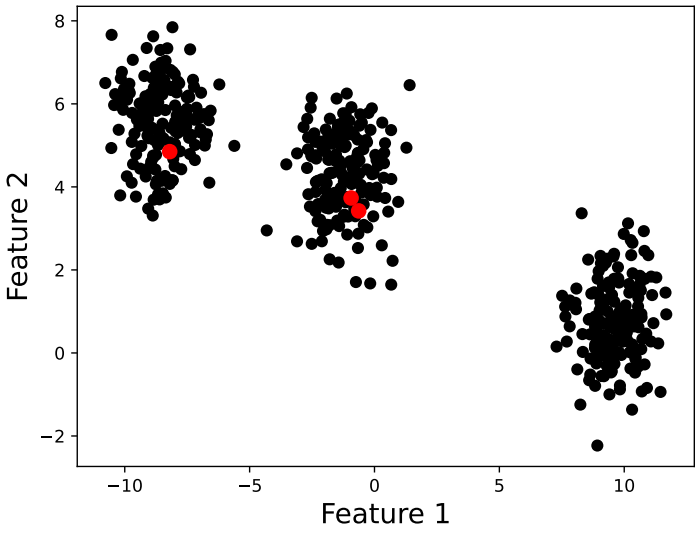
kMeans++ Example

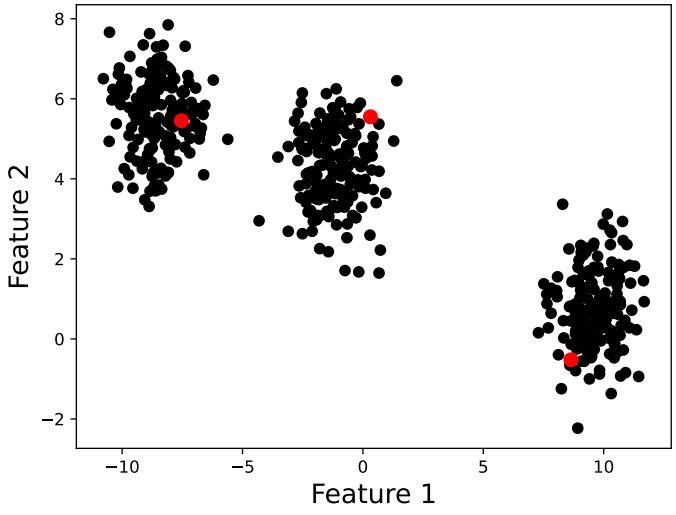


Random Initializations

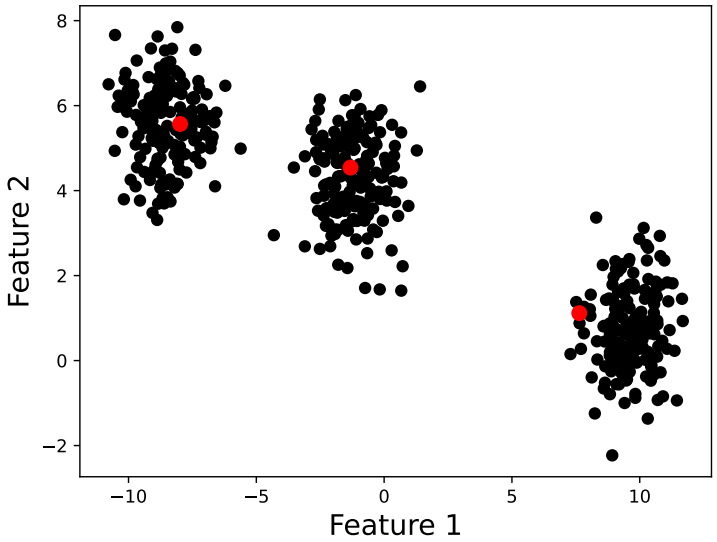


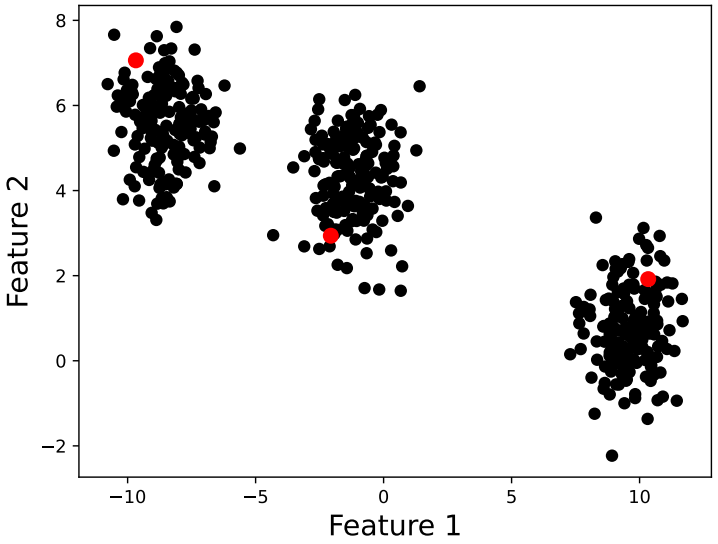


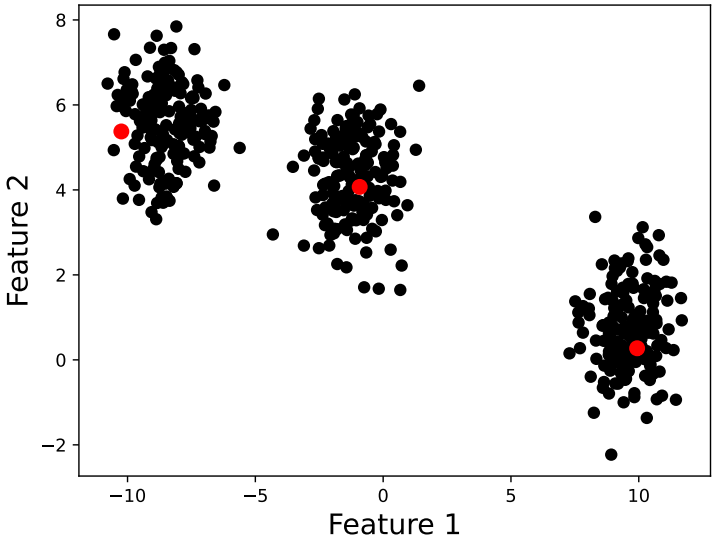


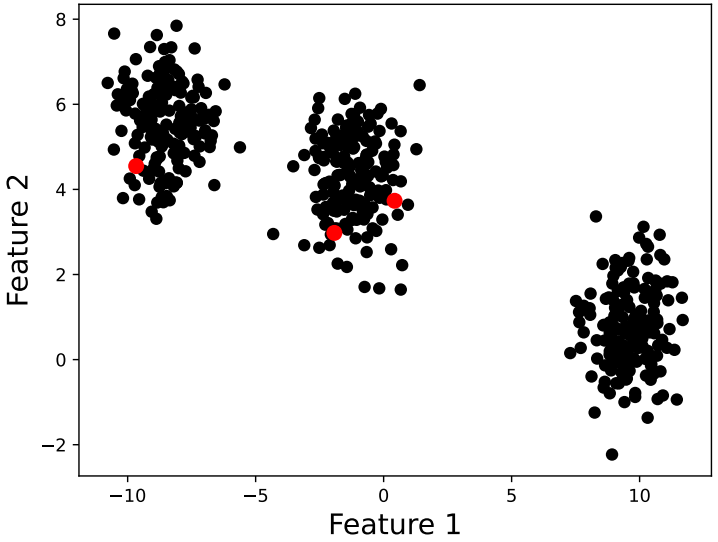


kMeans++ Initializations









Questions?

kMeans++

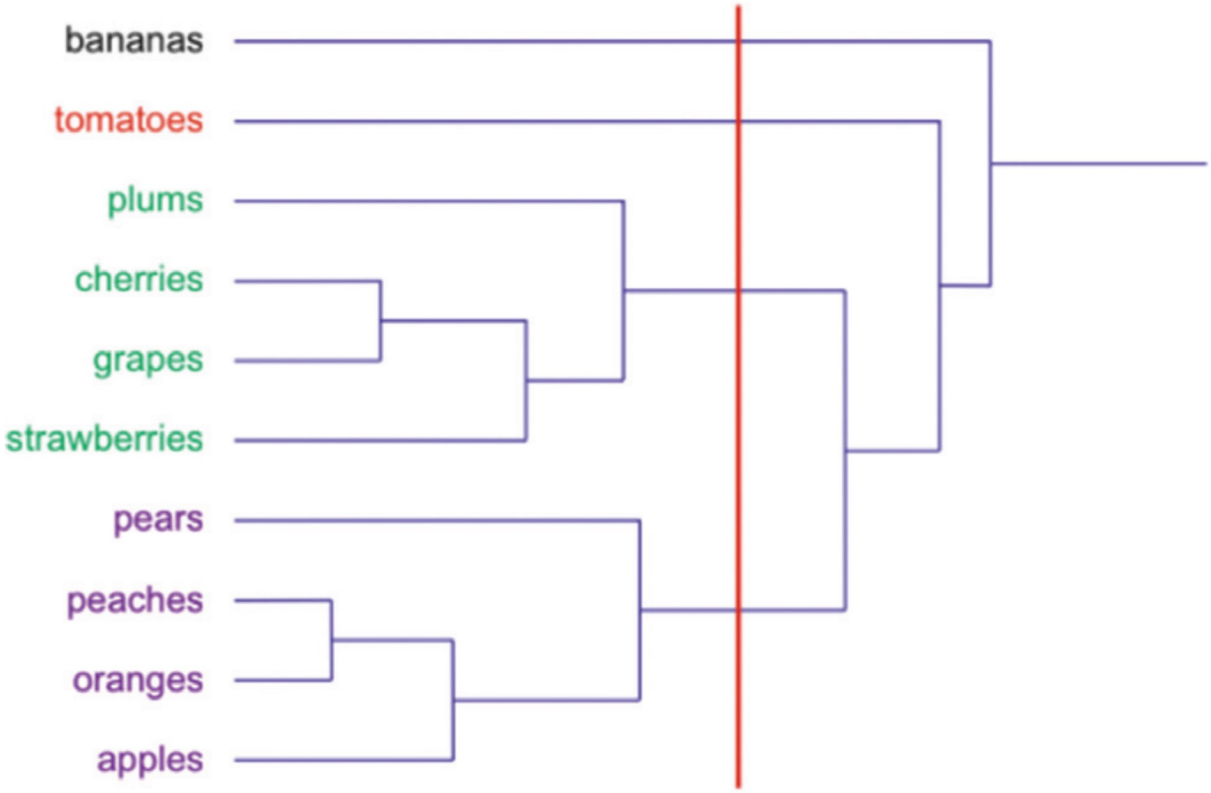
Hierarchical Clustering

Dimensionality Reduction

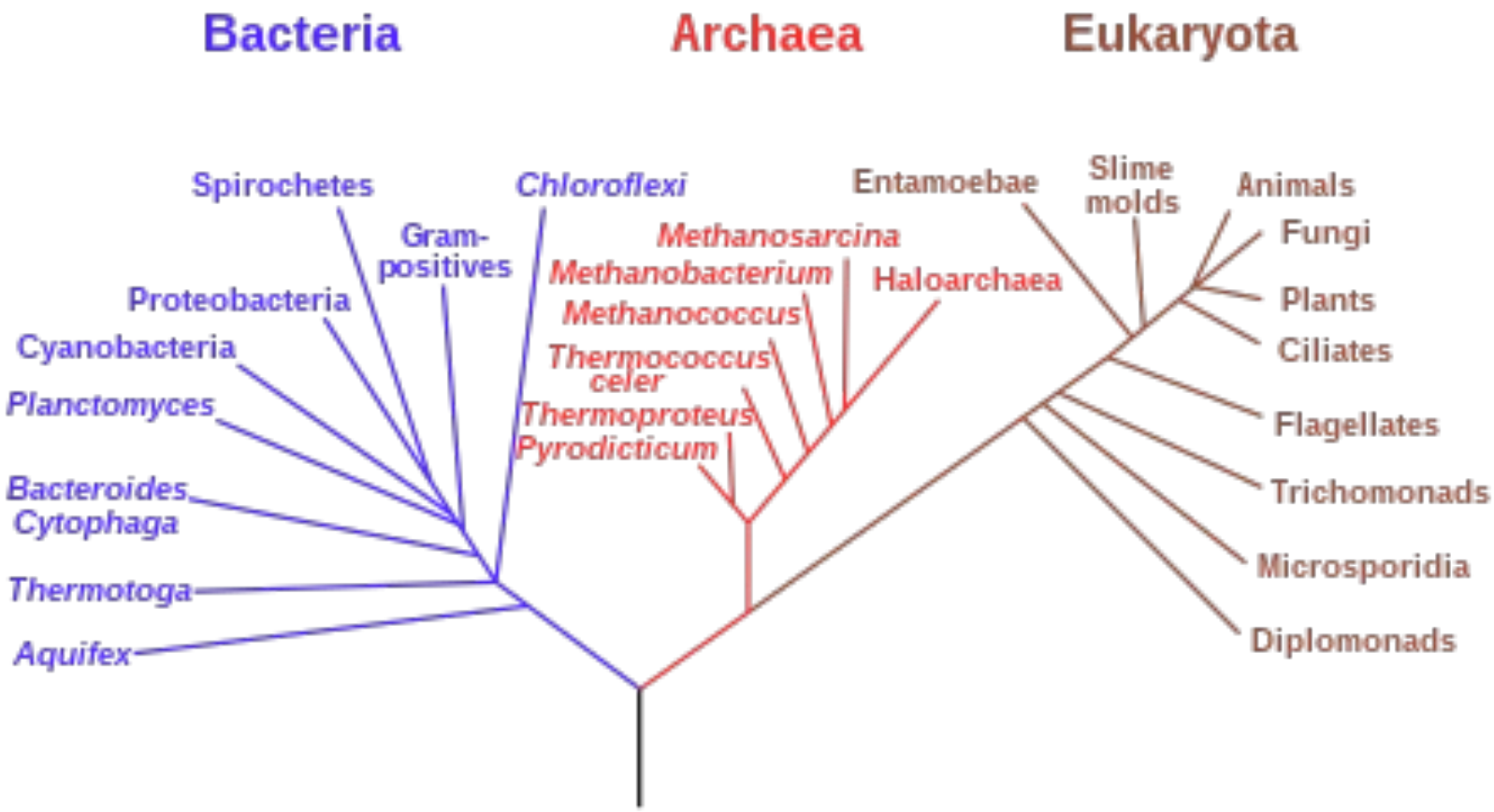
Hierarchical Clustering

General idea: build a tree of clusters, rather than a single clustering

Useful for exploratory analysis of smaller data sets



Hierarchical Clustering



Phylogenetic Trees

Hierarchical Clustering

General idea: build a tree of clusters, rather than a single clustering

Useful for exploratory analysis of smaller data sets

Different approaches:

1. **Agglomerative** or "bottom up" (most widely used)
   Starts by merging datapoints into clusters, continues until all points are in 1 cluster

2. **Divisive** or "top-down" (less widely used)

Agglomerative Clustering Methods

All methods use a notion of distance between clusters:

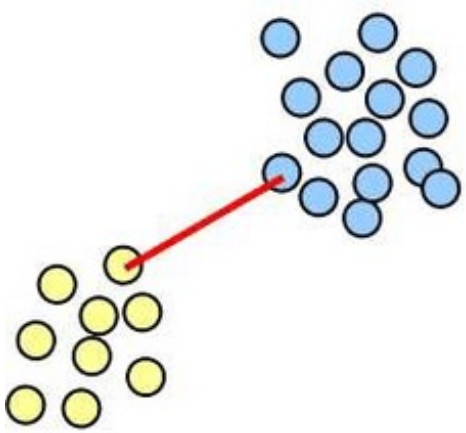
We will assume Euclidean distance (the usual default)
(but could use any distance measure, e.g., edit distance between strings)

Method 1: Nearest-Neighbor (or Single-Linkage)
-> merge the 2 clusters that has pair of datapoints from each cluster that are closest

Method 2: Furthest-Neighbor (or Complete-Linkage)
-> merge the 2 clusters that has pair of datapoints from each cluster that are furthest

Method 3: Average
-> merge the 2 clusters whose centroids are closest

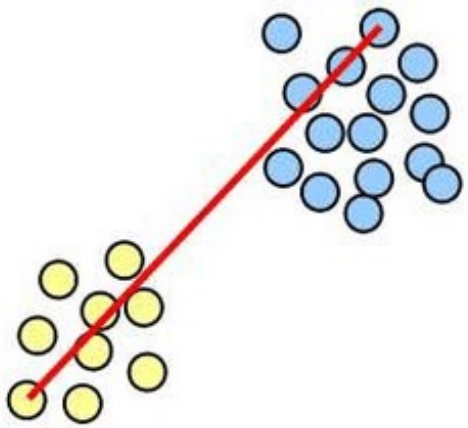
Visualizing Cluster-Merging Methods



single-link

(nearest neighbor)

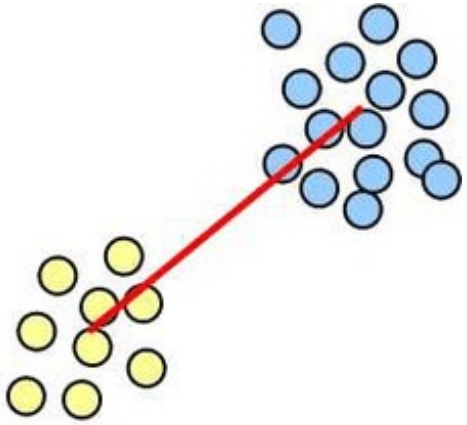
Can produce clusters with "elongated chain-like" shapes



complete-link

(furthest neighbor)

Compact-shaped clusters



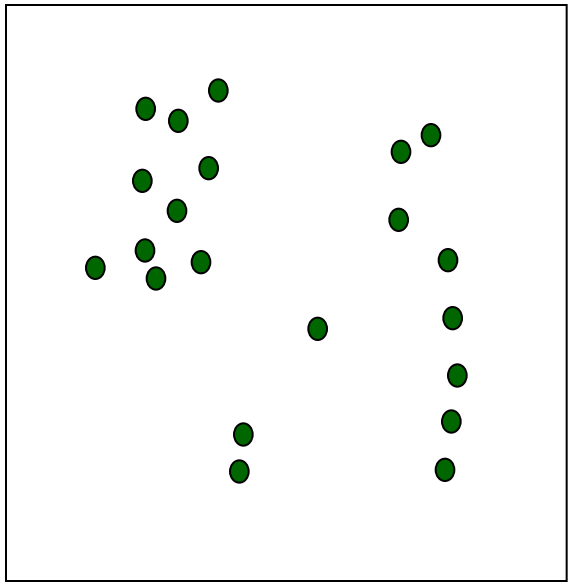
average-link

Compact-shaped clusters

Hierarchical Agglomerative Clustering

We will focus on single-link (nearest neighbor) clustering

Data:



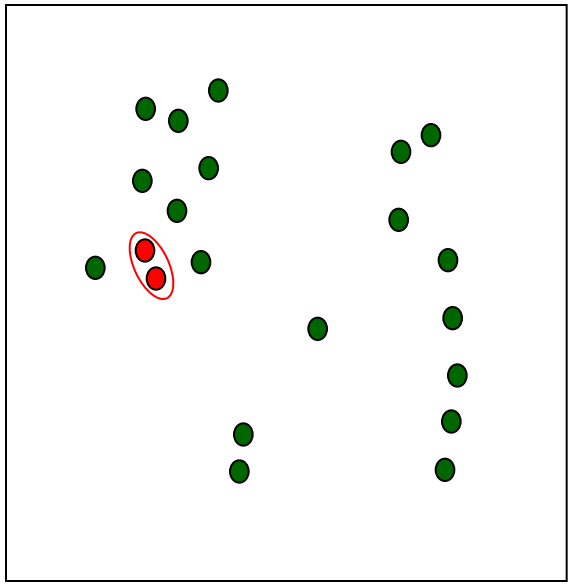
Initially:
- every point is its own cluster
- to begin:
  -> compute all pairwise distances
  -> find smallest distance

Figures adapted from Alex Ihler

Iteration 1

Merge the 2 closest clusters (all clusters are points at this stage)

Data:



Dendrogram:

Height of the join indicates dissimilarity

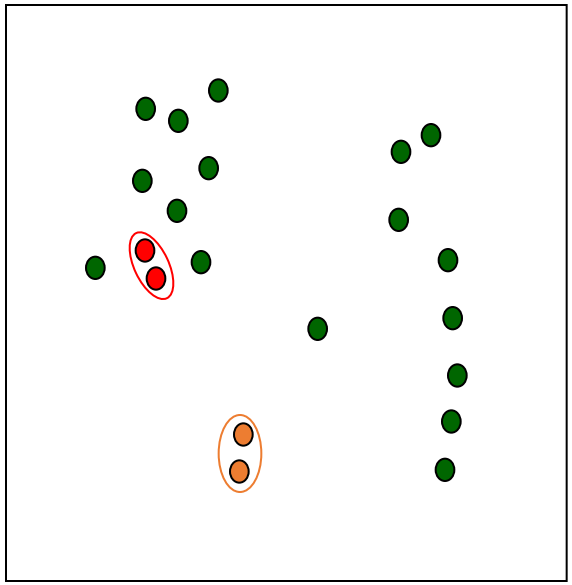
Sequentially build a hierarchy of clusters

Figures adapted from Alex Ihler

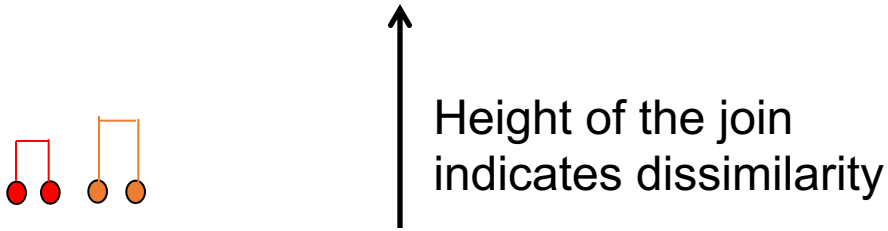
Iteration 2

Merge the 2 closest clusters (where clusters are 1 or 2 points)

Data:



Dendrogram:



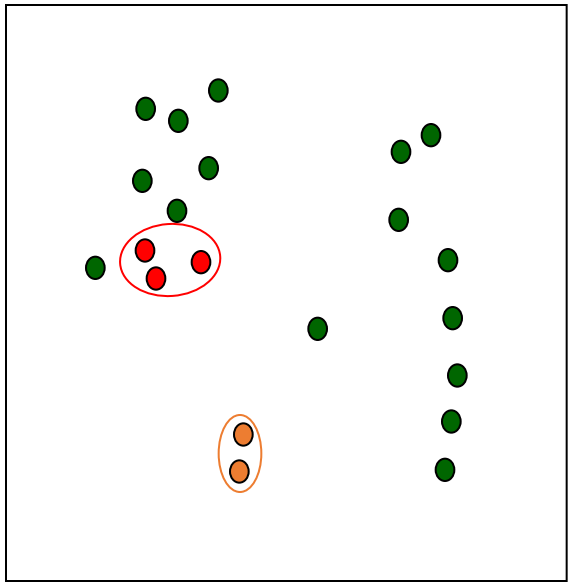
Sequentially build a hierarchy of clusters

Figures adapted from Alex Ihler

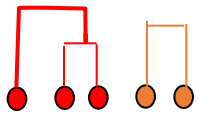
Iteration 3

Merge the 2 closest clusters ..and now one cluster gets 3 points

Data:



Dendrogram:



Height of the join indicates dissimilarity

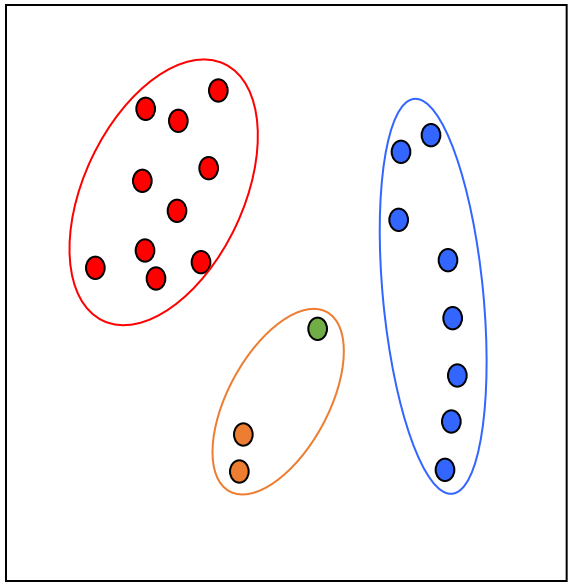
Sequentially build a hierarchy of clusters

Figures adapted from Alex Ihler

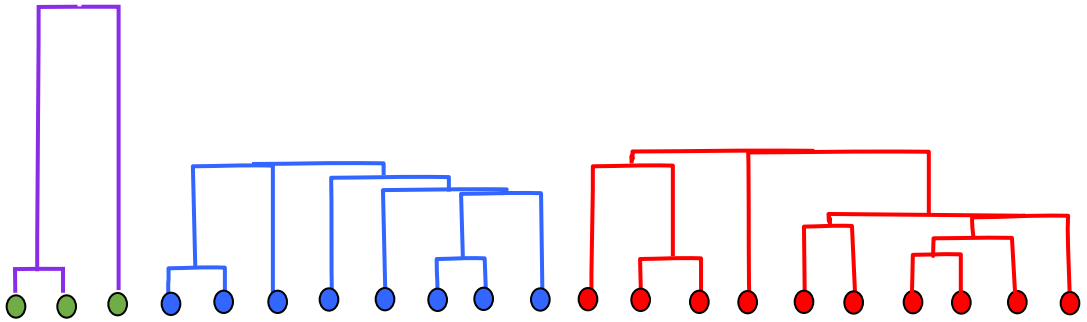
Iteration n-3

At this stage, only 1 individual point left (in green)…gets merged with orange

Data:



Dendrogram:



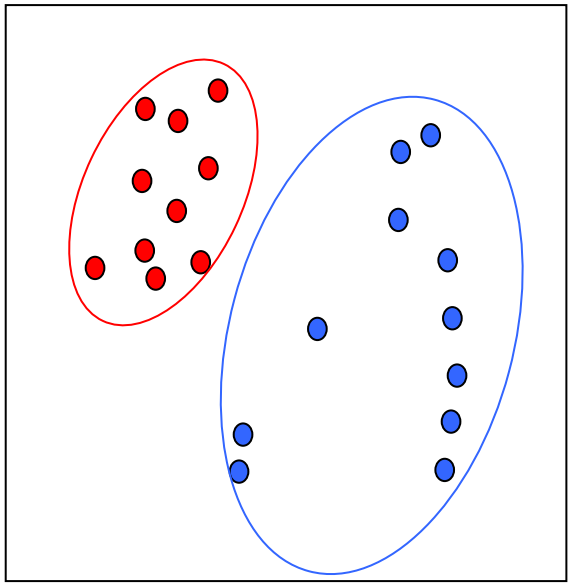
Sequentially build a hierarchy of clusters

Figures adapted from Alex Ihler

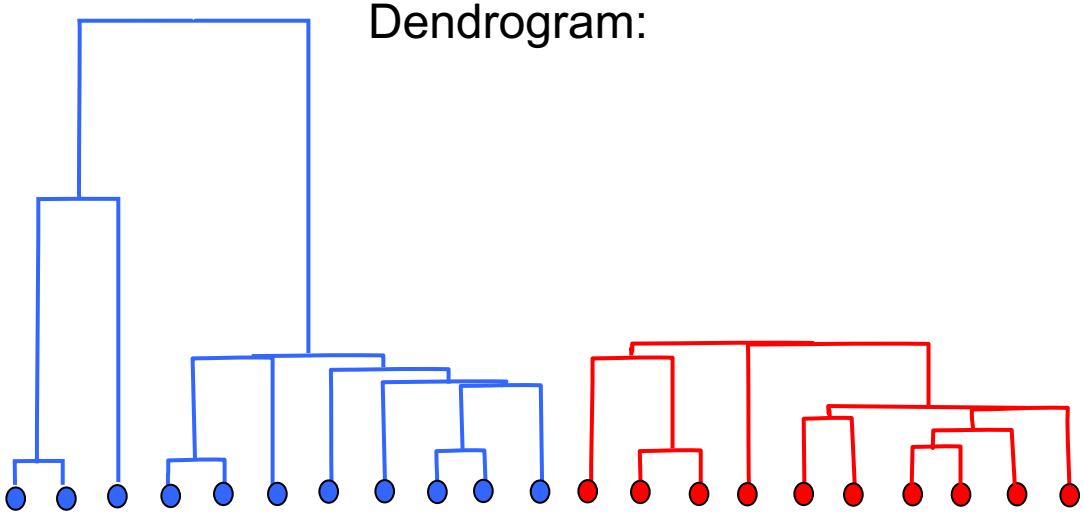
Iteration n-2

Blue and orange clusters get merged

Data:



Dendrogram:

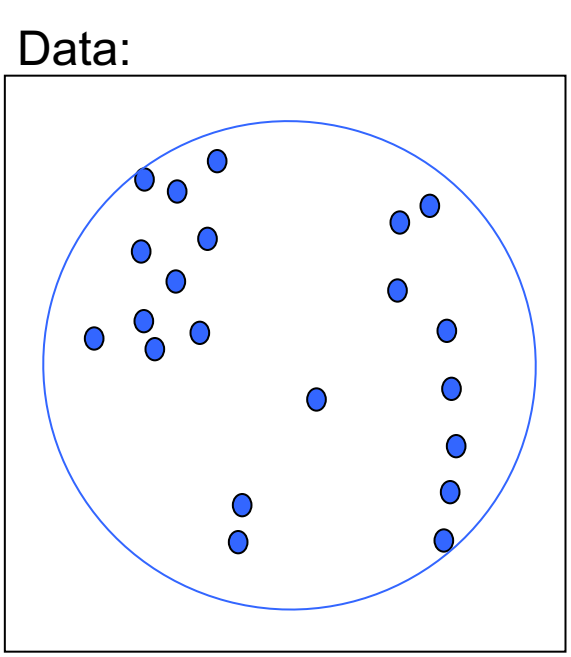


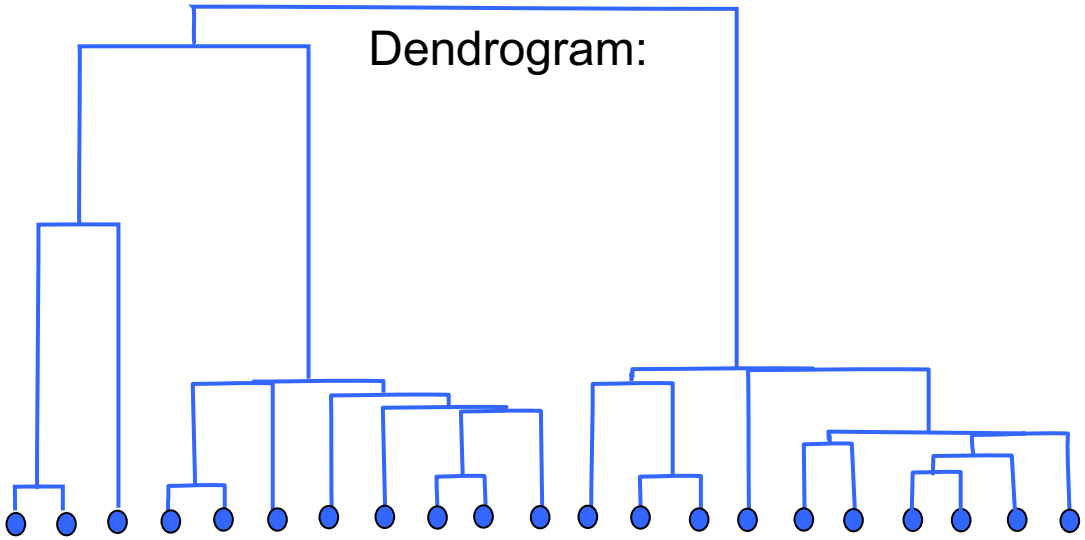
Sequentially build a hierarchy of clusters

Figures adapted from Alex Ihler

Iteration n-1

At the final iteration (n-1) all clusters/points are merged into 1 large cluster



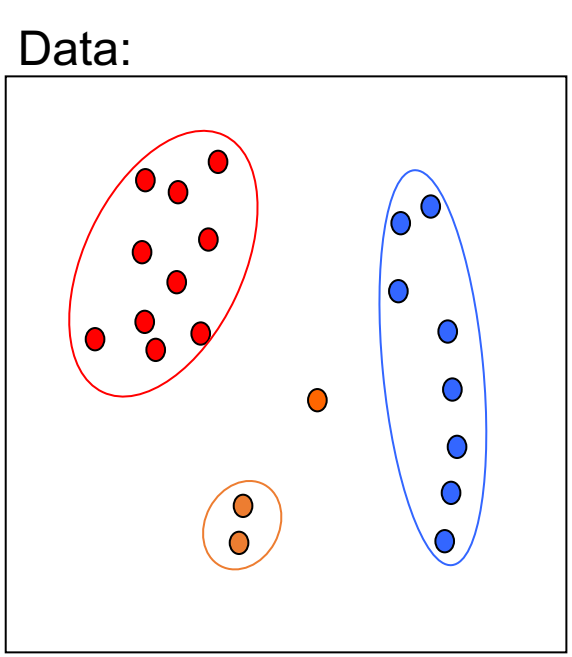


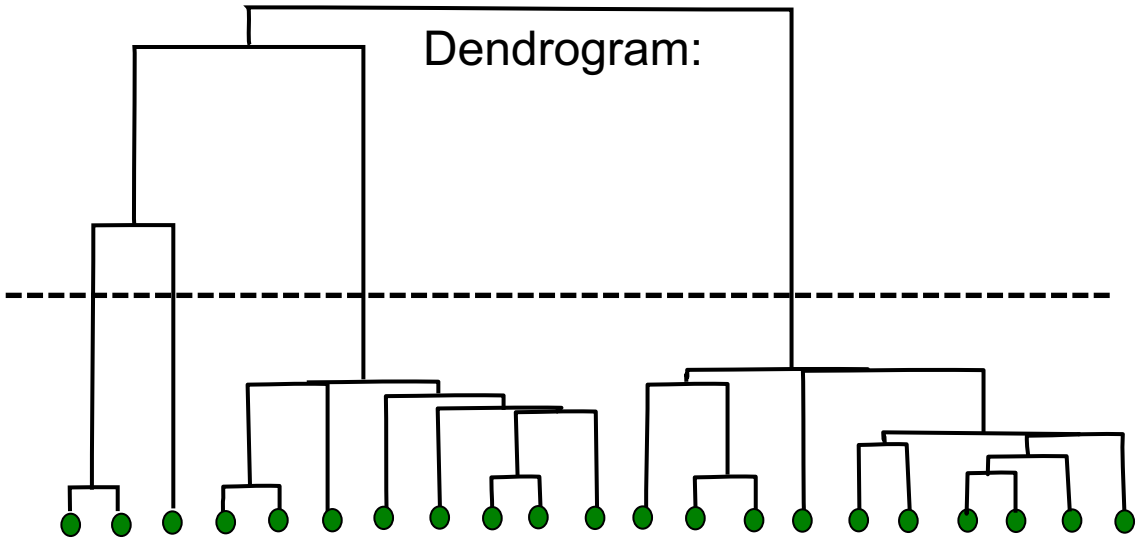
Sequentially build a hierarchy of clusters

Figures adapted from Alex Ihler

From Dendrogram to Clusters

Given the sequence, we can select a specific clustering by "cutting the tree"





Figures adapted from Alex Ihler

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`.

affinity : *str or callable, default='euclidean'*
Metric used to compute the linkage. Can be "euclidean", "l1", "l2", "manhattan", "cosine", or "precomputed". If linkage is "ward", only "euclidean" is accepted. If "precomputed", a distance matrix (instead of a similarity matrix) is needed as input for the fit method.

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`.

Hierarchical Clustering of the Iris Dataset

(colors indicate true labels)

Generated using complete linking method

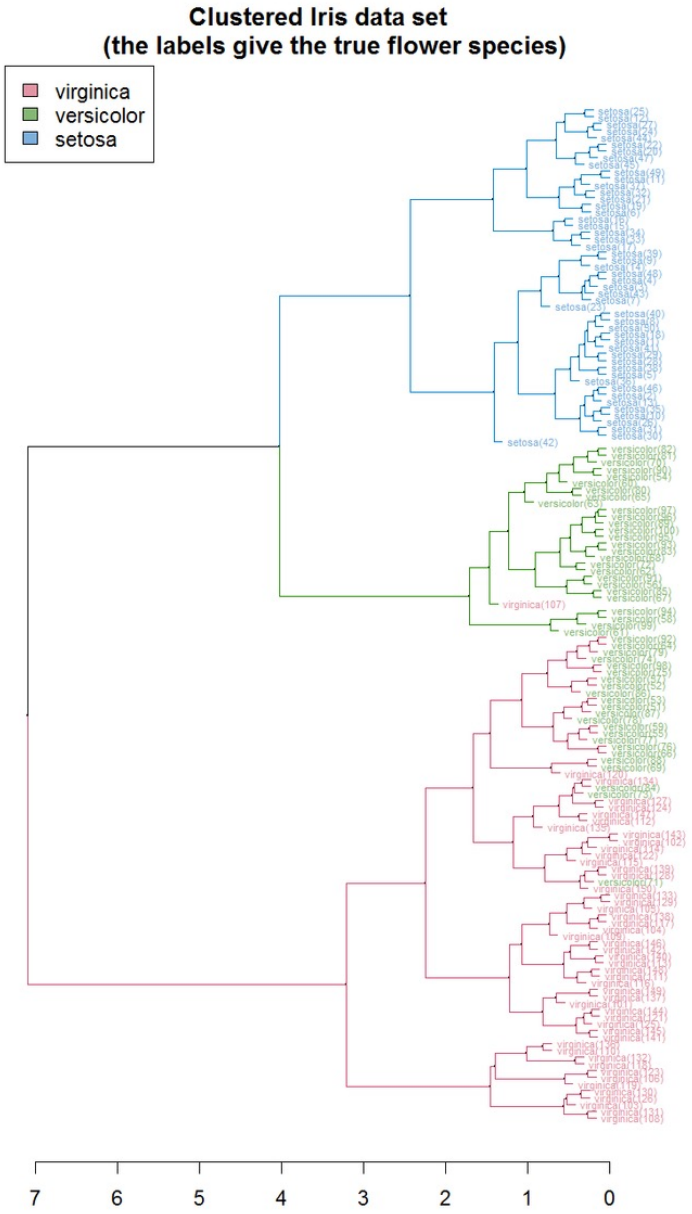
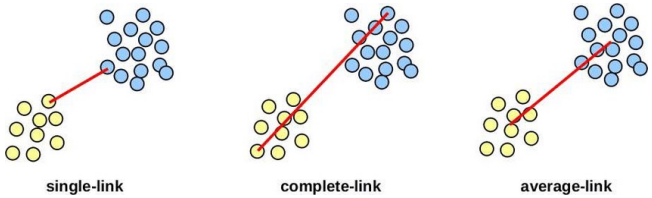


Figure from https://cran.r-project.org/web/packages/dendextend/vignettes/Cluster_Analysis.html

Type of Cluster Distance affects Cluster Shapes



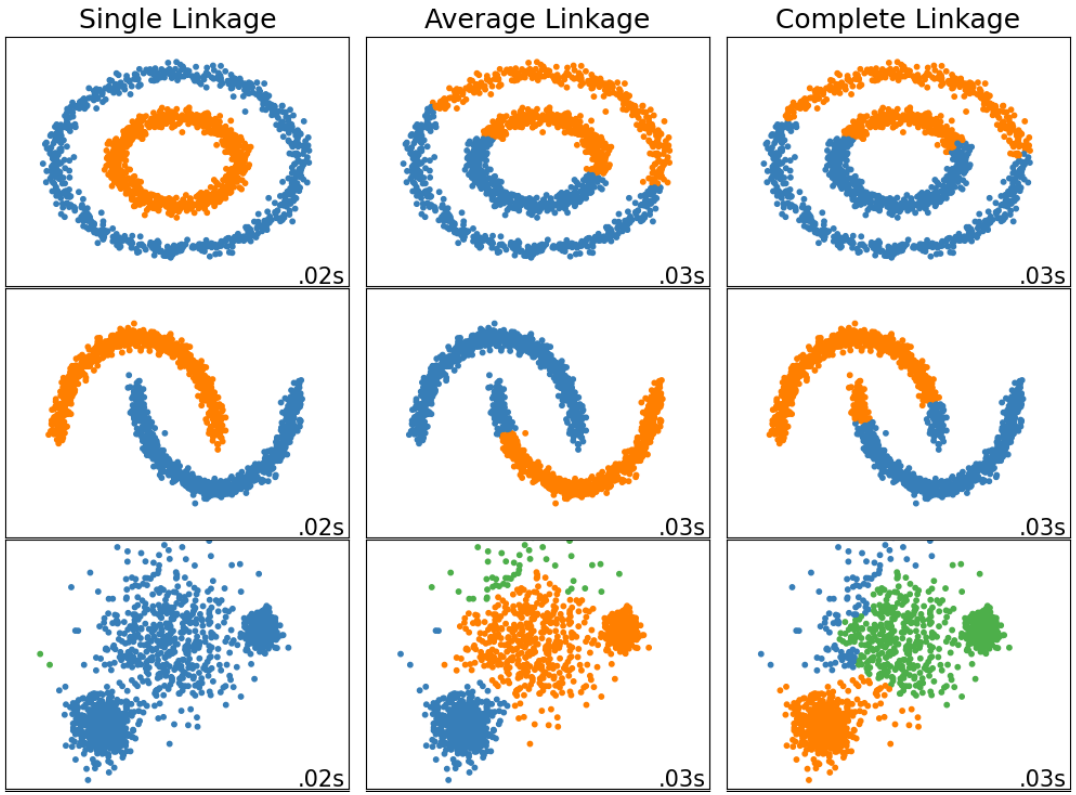
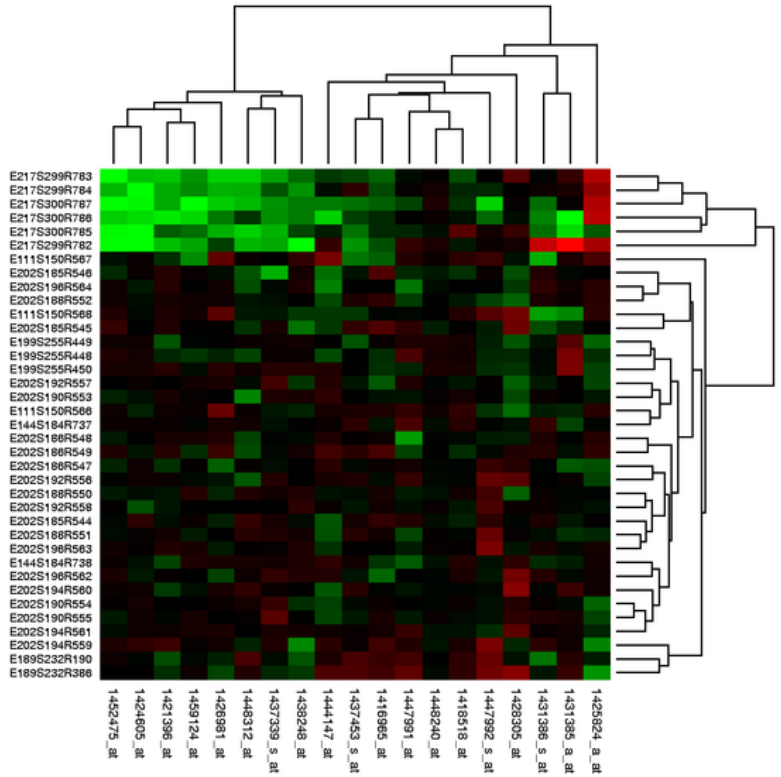


Figure from
https://scikit-learn.org/stable/auto_examples/cluster/plot_linkage_comparison.html

Example: Gene Expression Data in Biology

- Data (on right):
  - Rows = patients, columns = genes
  - Color = gene expression (activity)
- Biological discovery
  - What genes change together?
  - Are there clusters of patients
- Can cluster both genes or patients



Time Complexity of Hierarchical Clustering

Depends on the details of implementation and distance….

…. generally will be between $O(d\ n^2)$ and $O(d\ n^3)$

$O(d\ n^2)$ to initially compute all pairwise d-dimensional distances

n-1 iterations: could recompute all distances each time => $O(d\ n^2\ n) = O(d\ n^3)$

But can usually cache distance calculations and avoid n^3

However, the n^2 dependency (at least) means that hierarchical clustering is generally best suited for relatively small datasets (e.g., $n < 1000$)

Summary of Clustering Methods

- Clustering:
  - Broadly useful method for automatically finding groups in data
  - Part of “unsupervised learning”

- K-Means Clustering
  - Simple algorithm, minimizes squared error
  - Will produce “compact” clusters
  - Can converge to local minima, sensitive to initialization
  - Scales linearly in n

- Hierarchical Agglomerative Clustering
  - Broad family of methods
  - Can find “non-compact” clusters (with single-link)
  - Scales at least quadratically in n

Questions?

kMeans++

Hierarchical Clustering

Dimensionality Reduction

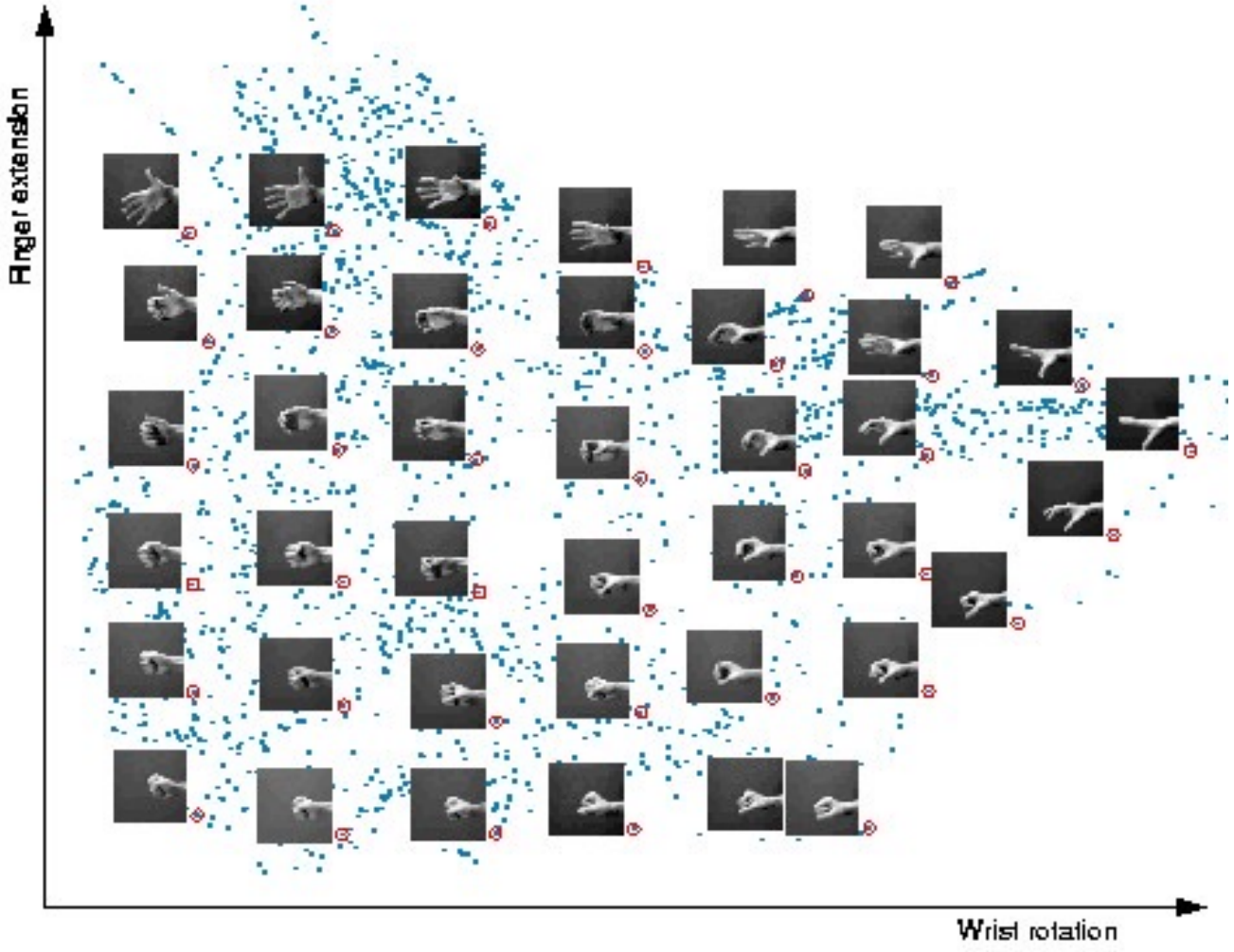
Motivation

- High-dimensional data
  - Images of faces (e.g. 28x28 pixels)
  - Text from articles (e.g. 10K different words)
  - All S&P 500 stocks (e.g. 500 stocks)

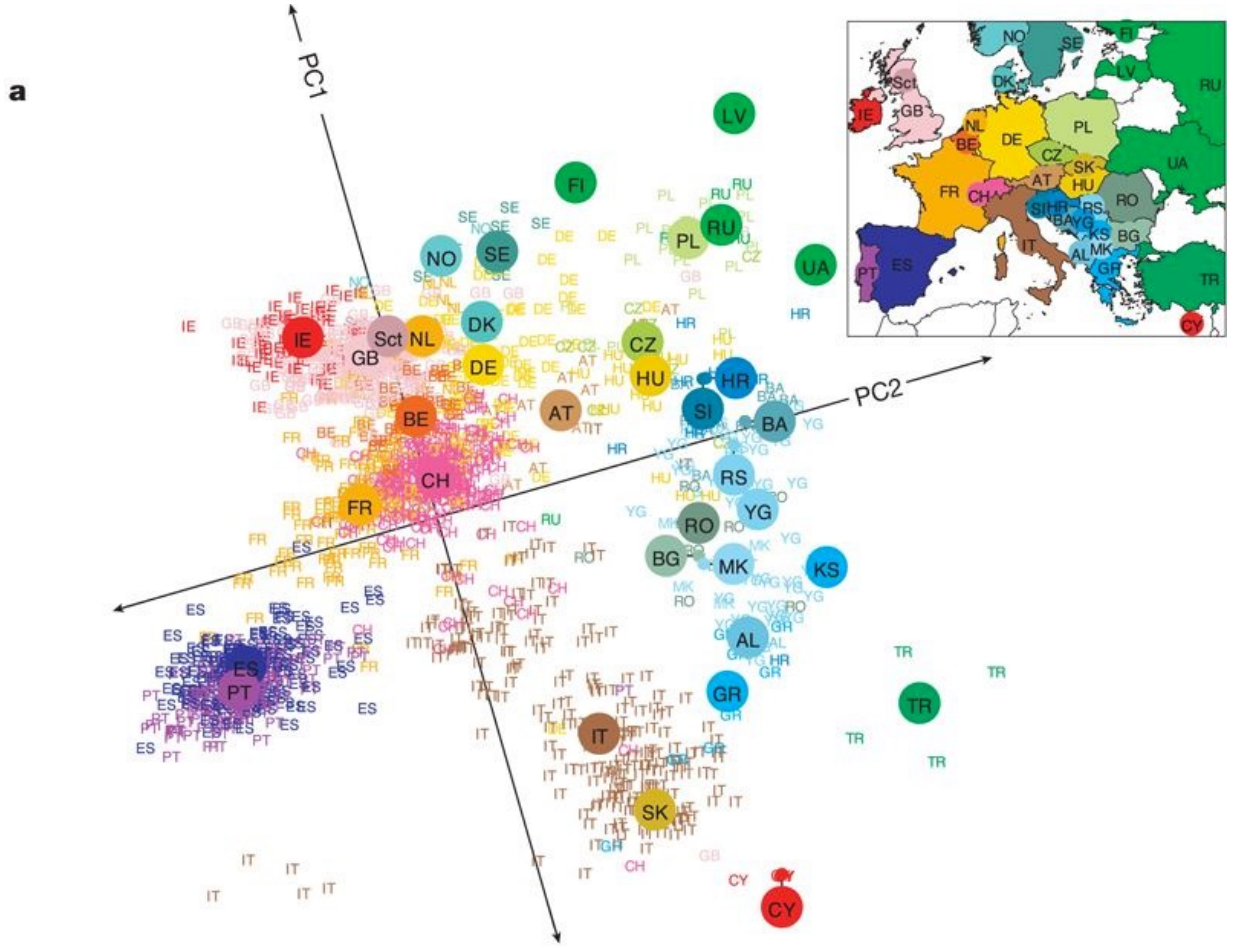
- Can we describe them in a simpler way?
  - Embedding: map data to a lower dimensional space
  - such that “similar” data are close

- Data often has a “low dimensional structure” we can leverage

Embedding Hand Poses

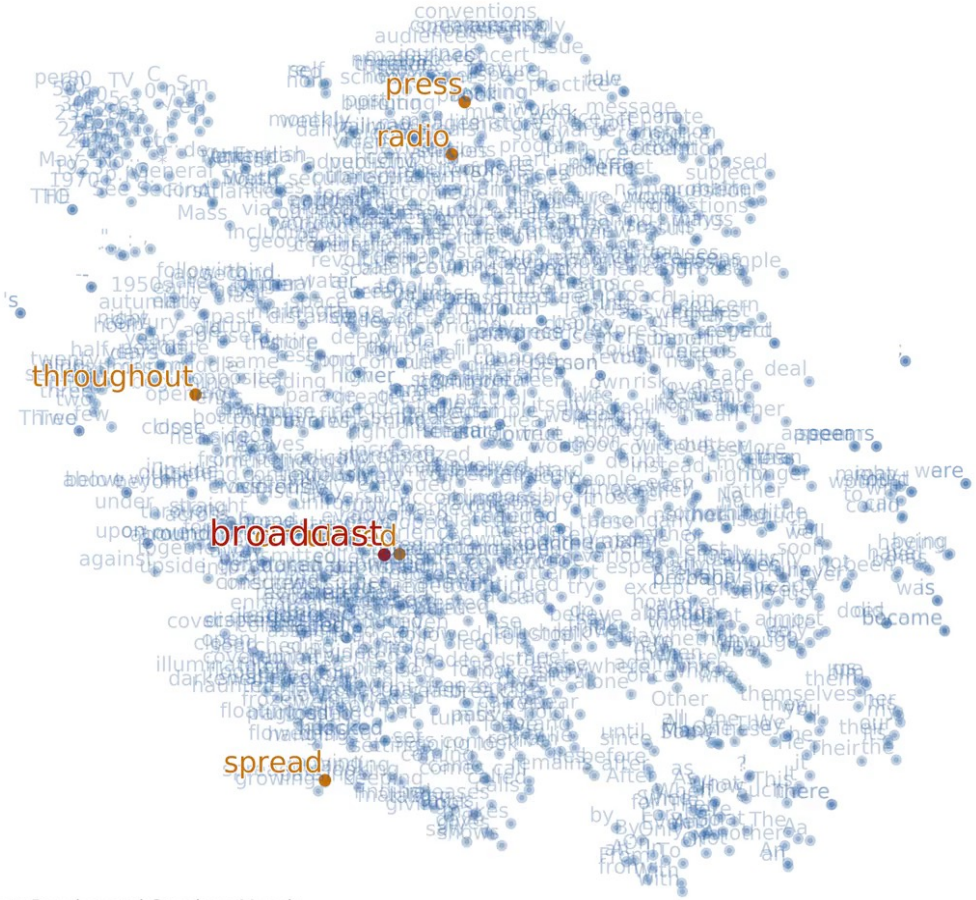


Genes mirror geography within Europe



Novembre et al., Nature 2008

Dynamic Embeddings of Words



1932

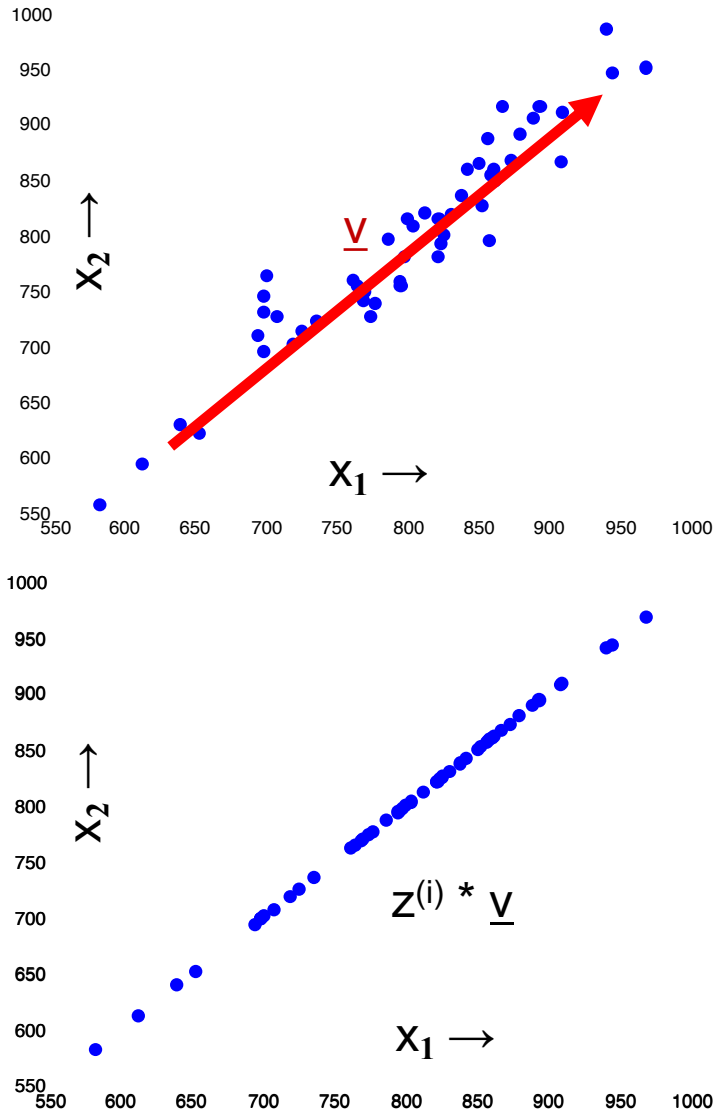
broadcast

radio
circulated
throughout
press
spread

Robert Bamler and Stephan Mandt,
Dynamic Word Embeddings, ICML 2017.

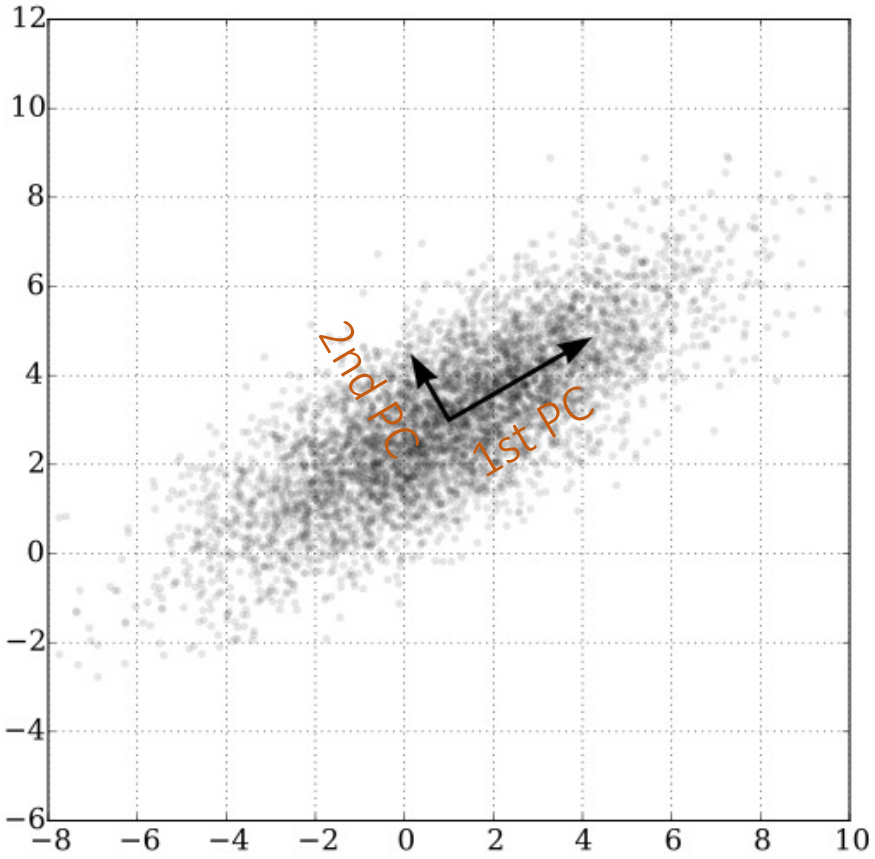
Dimensionality reduction

- Ex: Data with *two* real features (x_1,x_2)
  - Goal: Describe using only *one* value z
- We'll use a "model" to convert:
  - (x_1, x_2) ≈ f(z)
- Ex: linear function f(z):
  - f(z) = z * v = z * (v_1,v_2)
- v is the same for all data points
- z*v tells us the closest point to (x_1,x_2) along v



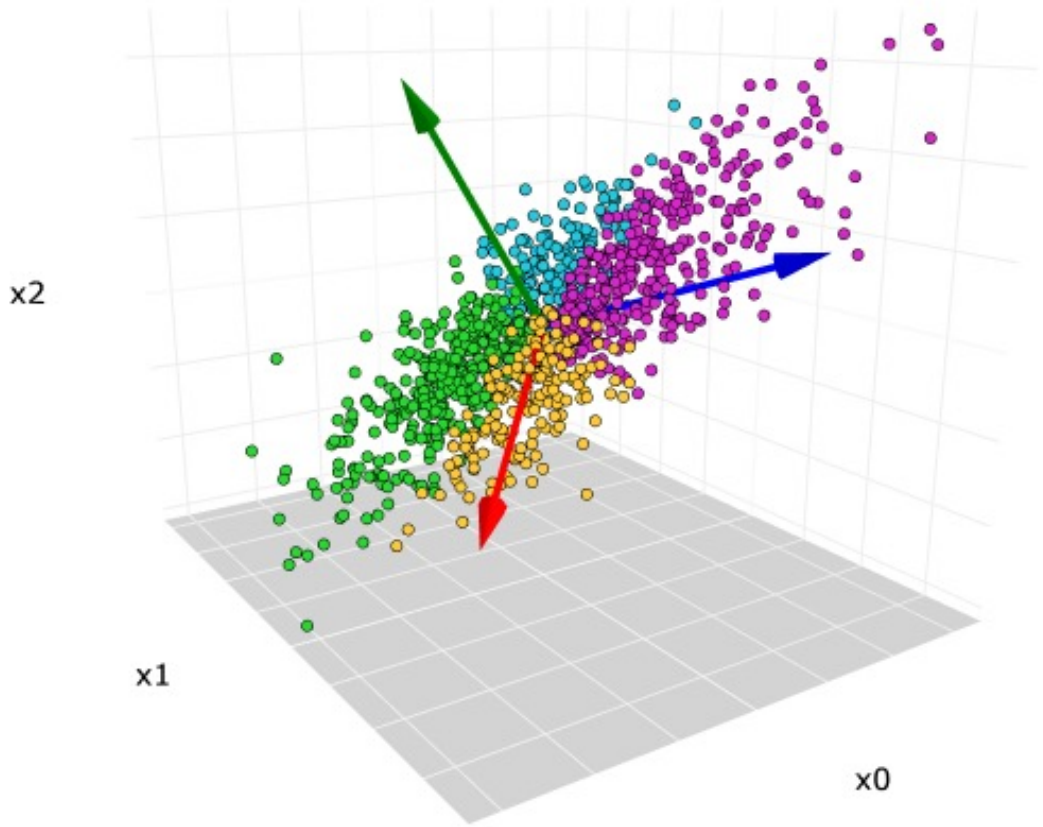
Principal Components Analysis (PCA)

Find the “directions” in the data space along which the data varies



Principal Components Analysis (PCA)

Find the “directions” in the data space along which the data varies



Principal Components Analysis (PCA)

Find the "directions" in the data space along which the data varies

Each principal component is a d dimensional vector
(i.e. same dimensionality of the data)

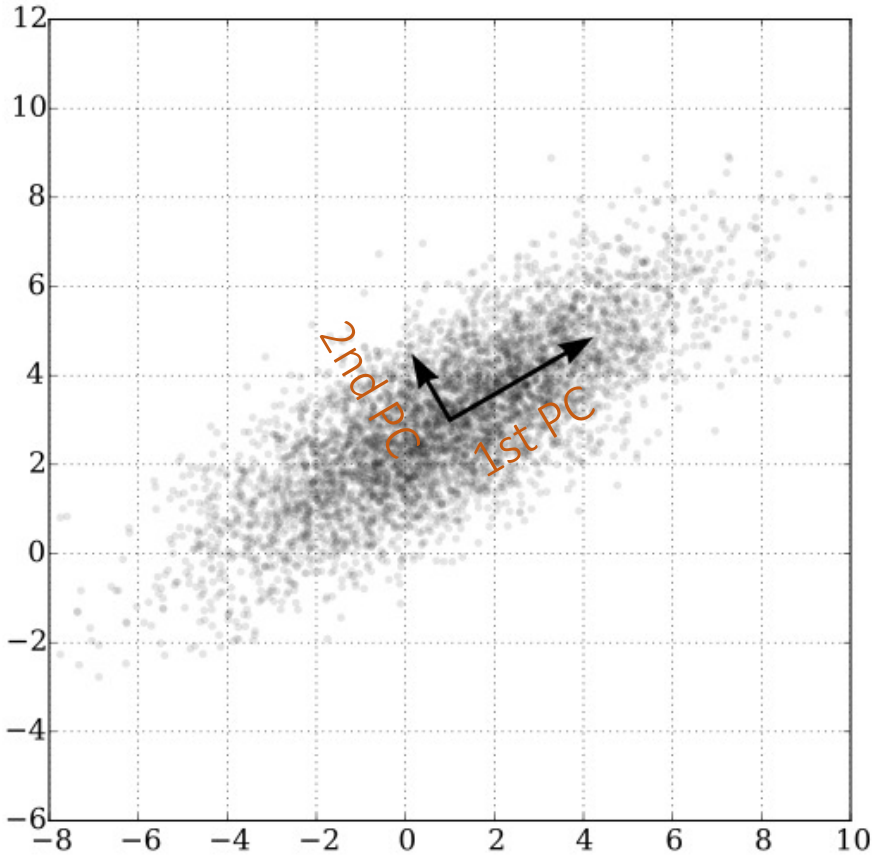
Principal components are *ordered*

- First principal component is the direction that best explains the variance in the data
- Second is the direction that:
  1. Is orthogonal (perpendicular) to the first principal component
  2. Best explains the remaining variance
- And so on…

Can have at most d principal components
Principal components assumed to be normalized, i.e. have length one

Principal Components Analysis (PCA)

Find the “directions” in the data space along which the data varies



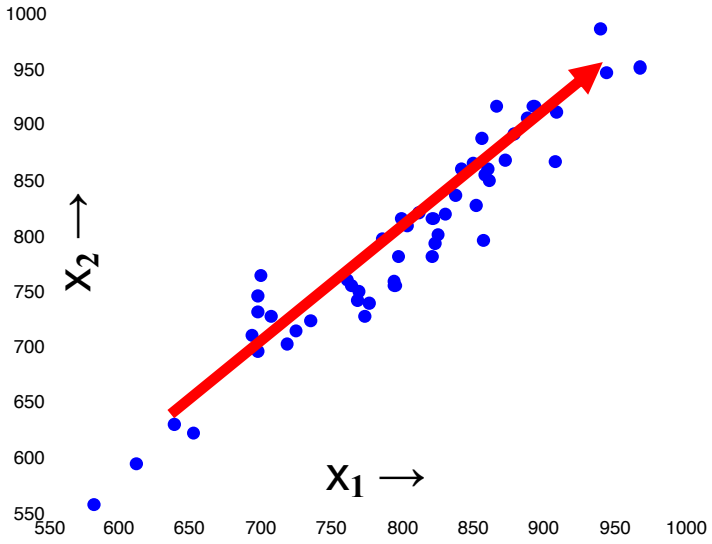
Dimensionality Reduction with PCA

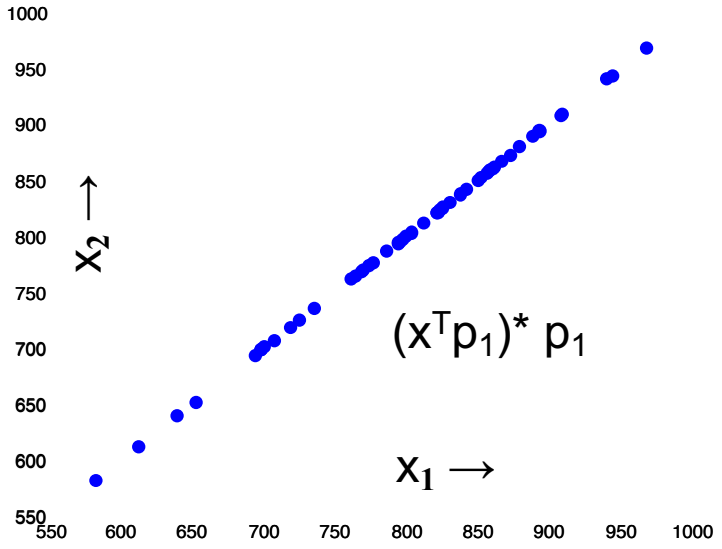
Given a list of ordered principal components p_1, p_2, …, p_M
Dimensionality reduction is performed by *projecting* data onto the space spanned by the first few principal components

Example: project **x** onto first principal component

$$x \rightarrow (x^T p_1) \cdot p_1$$

Only need a single number $(x^T p_1)$ to represent data!





Dimensionality Reduction with PCA

Given a list of ordered principal components p_1, p_2, …, p_M

Dimensionality reduction is performed by *projecting* data onto the space spanned by the first few principal components

Example: project x onto first two principal components

$$x \rightarrow (x^T p_1) \cdot p_1 + (x^T p_2) \cdot p_2$$

Now only need two numbers $(x^T p_1, x^T p_2)$ to represent data

In general, projection onto first M principal components

$$x \rightarrow \sum_{j=1}^{M} (x^T p_j) \cdot p_j$$

Finding Principal Components

Full details beyond the scope of this course

Basic steps:

- Center the data: subtract the mean of each feature

$$\boldsymbol{X} \leftarrow \boldsymbol{X} - \boldsymbol{\mu}$$

- Compute the *covariance* matrix of the centered data

$$\boldsymbol{C} = \boldsymbol{X}^T \boldsymbol{X}$$

  - C is a dxd matrix where the entry C_{ij} is the covariance between feature i and feature j

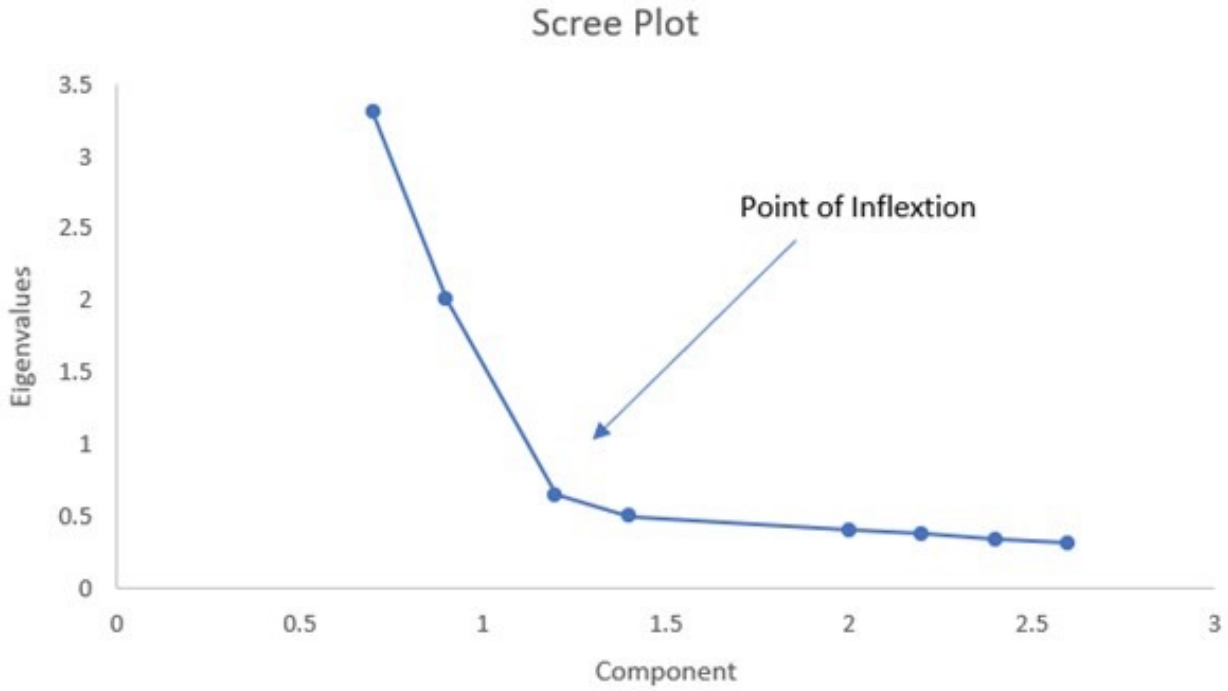
- The principal components are the *eigenvectors* of C
  - There exist many numerical routines for finding eigenvectors of a matrix

Finding Principal Components

```
mu = np.mean( X, axis=0, keepdims=True )   # find mean over data points
X0 = X - mu                # zero-center the data
S = X0.T.dot( X0 ) / m       # S = np.cov( X.T ), data covariance
D,V = np.linalg.eig( S )     # find eigenvalues/vectors: can be slow!
pi = np.argsort(D)[::-1]     # sort eigenvalues largest to smallest
D,V = D[pi], V[:,pi]         #
D,V = D[0:k], V[:,0:k]       # and keep the k largest
```

Finding Principal Components

How many principal components should we use?
Look at the corresponding *eigenvalues*



Questions?

Wrapup

- kMeans++
  - Alternative initialization strategy for kMeans
  - Initialize clusters sequentially based on distance to previous clusters
- Hierarchical Clustering
  - Builds a tree of clusters
- Dimensionality reduction
  - Reducing the number of variables needed to represent out data
  - Popular technique: PCA