CSE/STAT 416

Other clustering methods

Pemi Nguyen
Paul G. Allen School of Computer Science & Engineering
University of Washington

May 4, 2022

Define Clusters

In their simplest form, a **cluster** is defined by

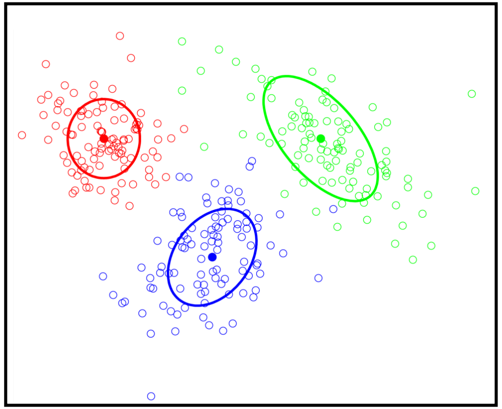
- The location of its center (**centroid**)
- Shape and size of its **spread**

Clustering is the process of finding these clusters and **assigning** each example to a particular cluster.

- x_i gets assigned $z_i \in [1, 2, \dots, k]$
- Usually based on closest centroid

Will define some kind of score for a clustering that determines how good the assignments are

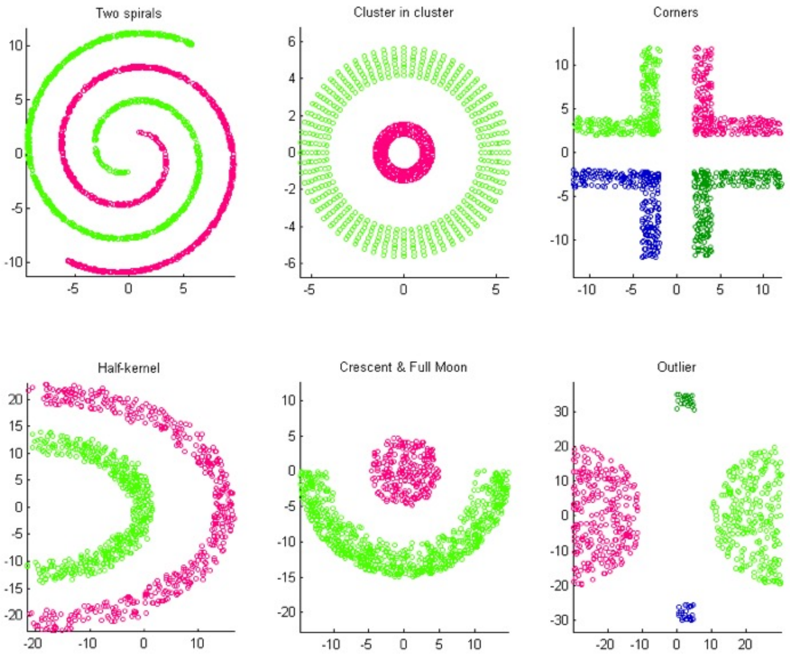
- Based on distance of assigned examples to each cluster



Not Always Easy

There are many clusters that are harder to learn with this setup

- Distance does not determine clusters



Smart Initializing w/ k-means++

Making sure the initialized centroids are "good" is critical to finding quality local optima. Our purely random approach was wasteful since it's very possible that initial centroids start close together.

Idea: Try to select a set of points farther away from each other.

k-means++ does a slightly smarter random initialization

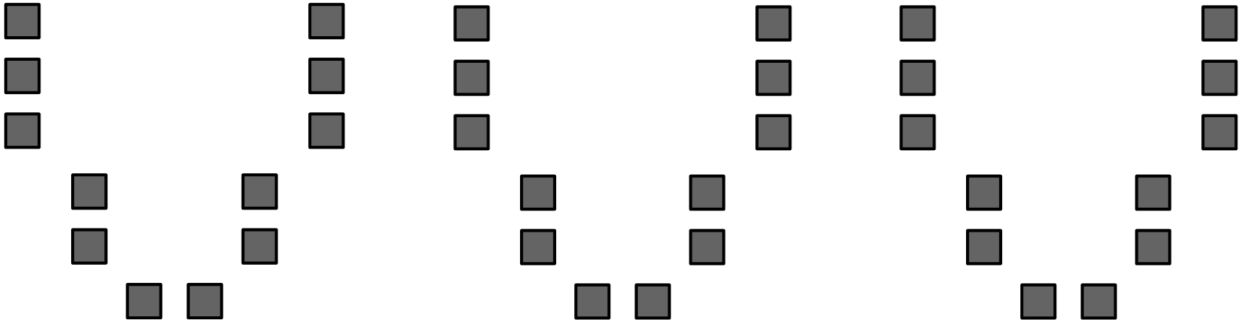
1. Choose first cluster $\mu^{(1)}$ from the data uniformly at random
2. For each data point $x^{(i)}$ not chosen yet, compute $D(x^{(i)})$, the distance between $x^{(i)}$ and the nearest centroid that has already been chosen.
3. Choose one new data point at random as a new centroid, using a weighted probability distribution where a point $x^{(i)}$ is chosen with probability proportional to $D\left(x^{(i)}\right)^2$.
4. Repeat 2 and 3 until we have selected k centroids

k-means++ Example

Start by picking a point at random

Then pick points proportional to their distances to their centroids

This tries to maximize the spread of the centroids!

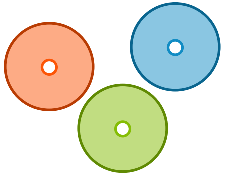


Problems with k-means

In real life, cluster assignments are not always clear cut

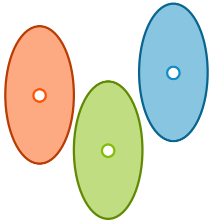
E.g. The moon landing: Science? World News? Conspiracy?

Because we minimize Euclidean distance, k-means assumes all the clusters are spherical



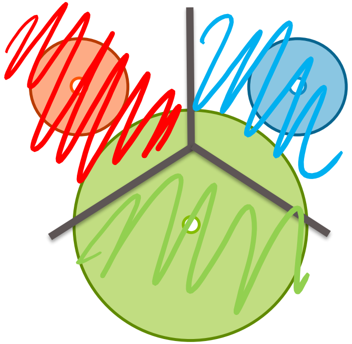
We can change this with weighted Euclidean distance

Still assumes every cluster is the same shape/orientation

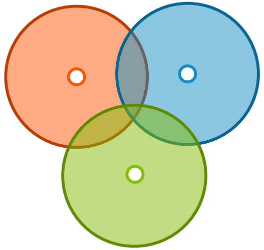


Failure Modes of k-means

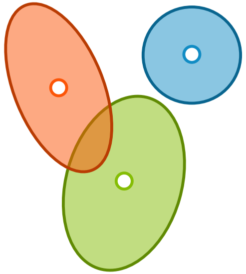
If we don't meet the assumption of spherical clusters, we will get unexpected results



disparate cluster sizes



overlapping clusters



different shaped/oriented clusters

Mixture Models

A much more flexible approach is modeling with a **mixture model**

Model each cluster as a different probability distribution and learn their parameters

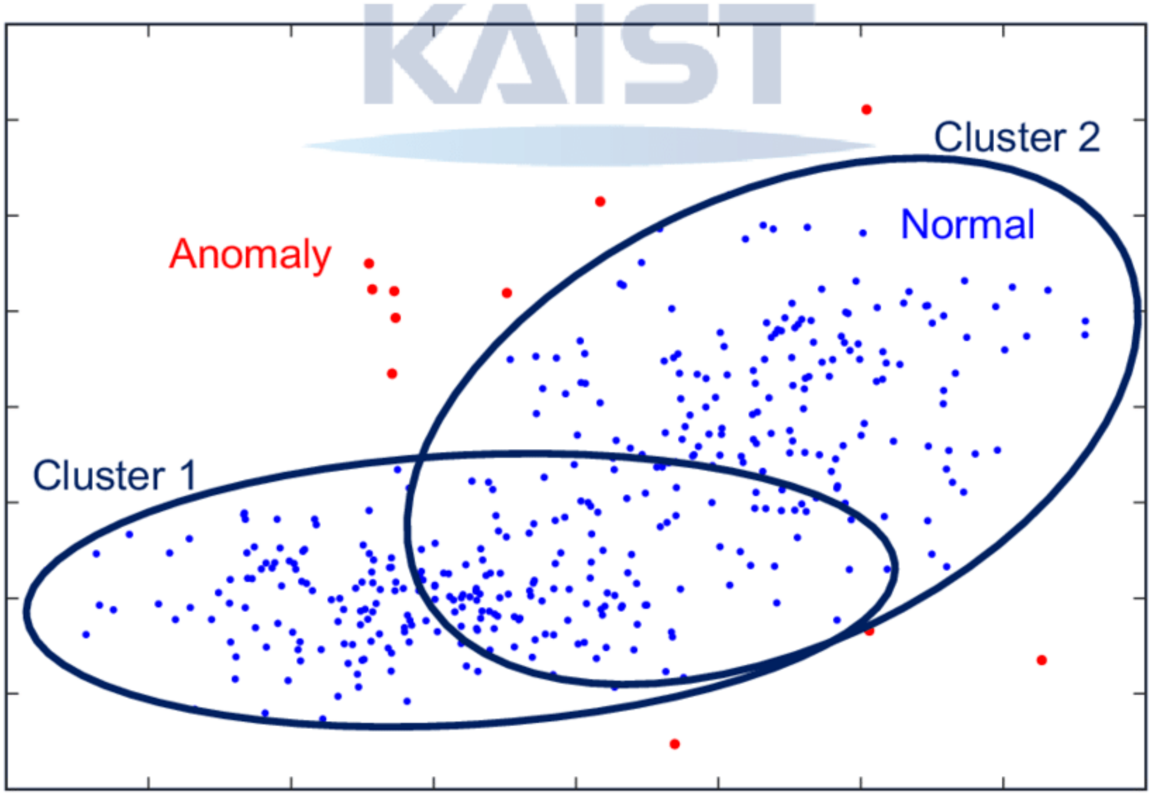
- One example is Gaussian Mixtures
- Allows for different cluster shapes and sizes
- Typically learned using Expectation Maximization (EM) algorithm

Allows **soft assignments** to clusters

- Example: A news article: 54% chance is about world news, 45% science, 1% conspiracy theory, 0% other

Gaussian Mixture Models

Anomaly Detection using Gaussian Mixture Models



https://www.researchgate.net/figure/Anomaly-detection-with-Gaussian-mixture-models_fig1_348675968

Two types of clustering

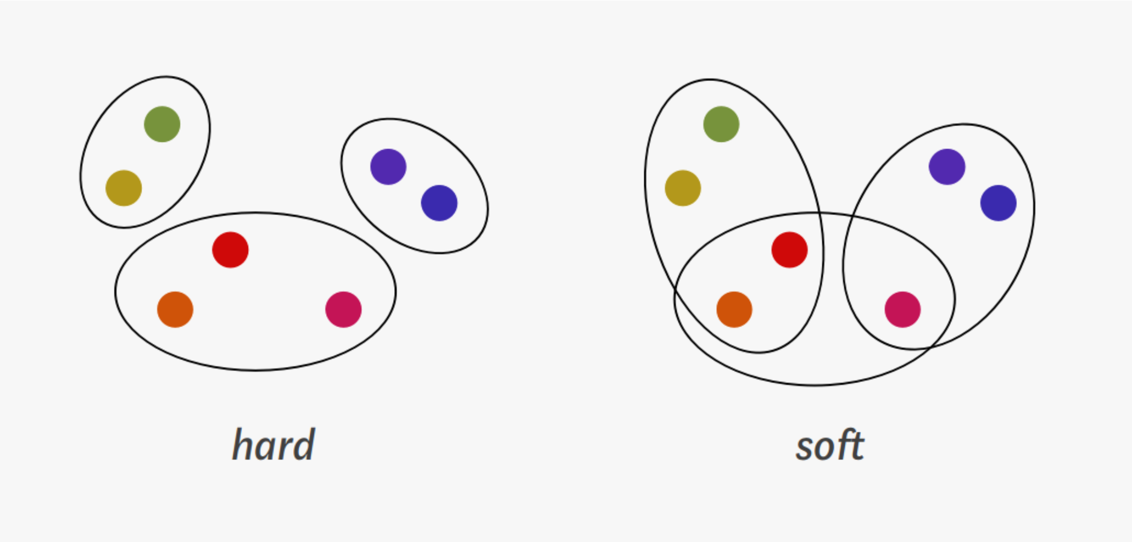
Hard clustering: clusters do not overlap

- E.g: K-means Clustering

Soft clustering: clusters **may** overlap

- E.g: Gaussian Mixtures

Note: Hard Clustering is a subset of Soft Clustering.



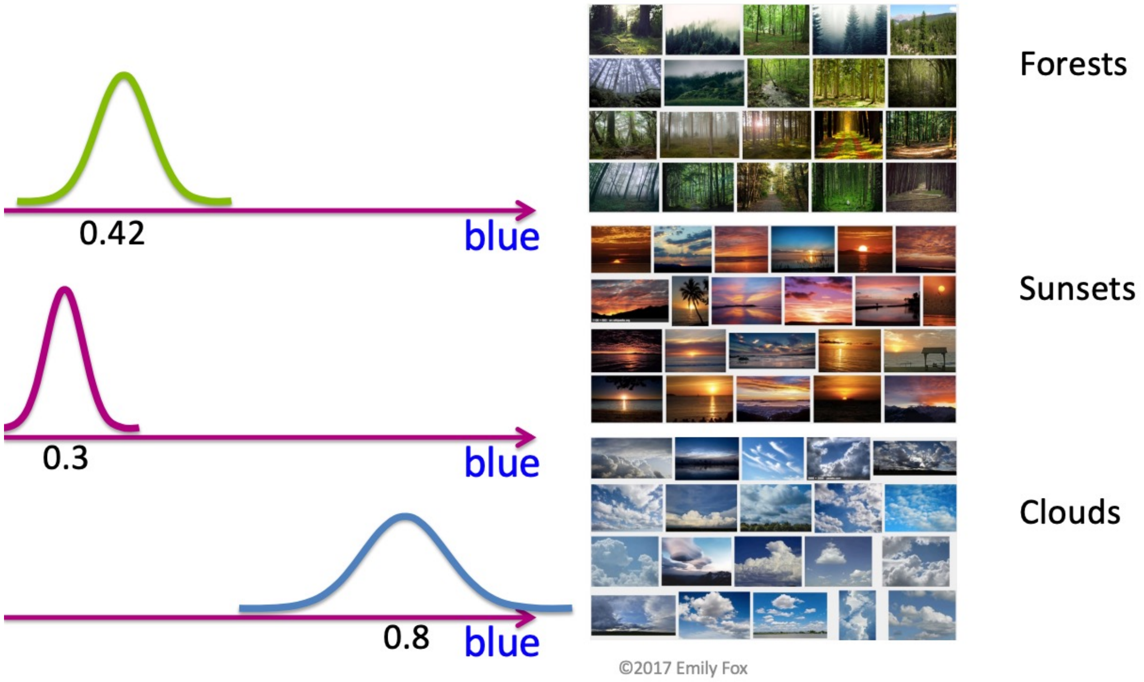
Mixture models

Probabilistically grounded way of clustering

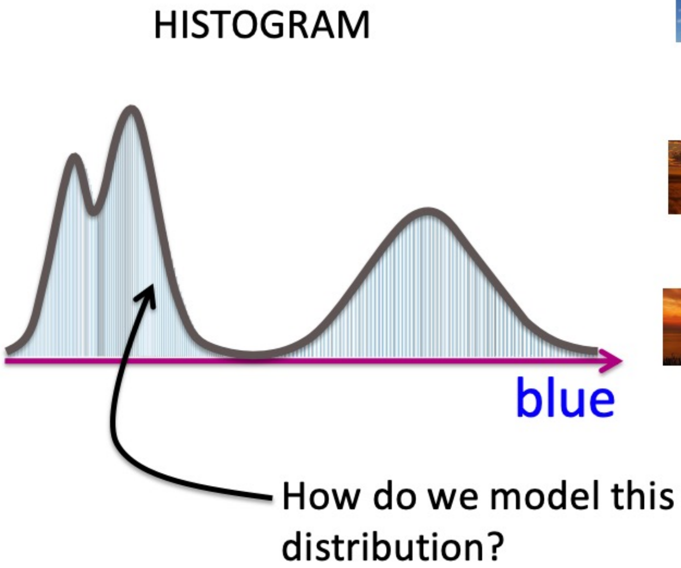
Each cluster is a generative model

The parameters are means and covariances of each cluster

Model as Gaussian per cluster

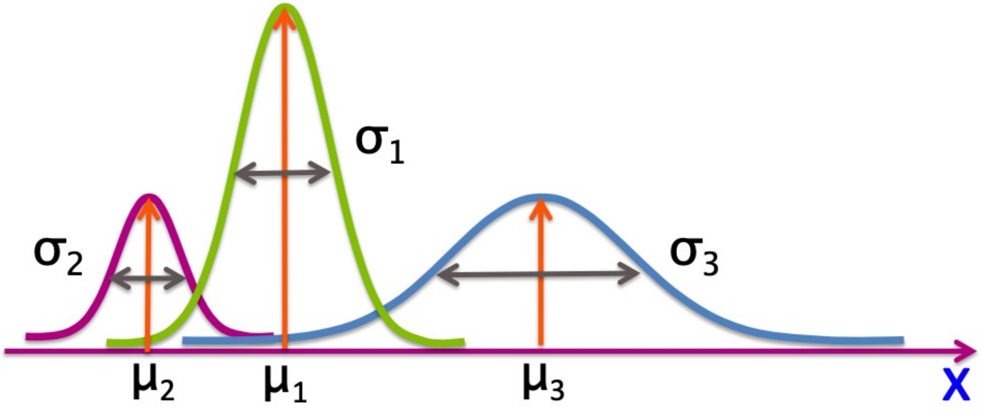


Model as Gaussian per cluster



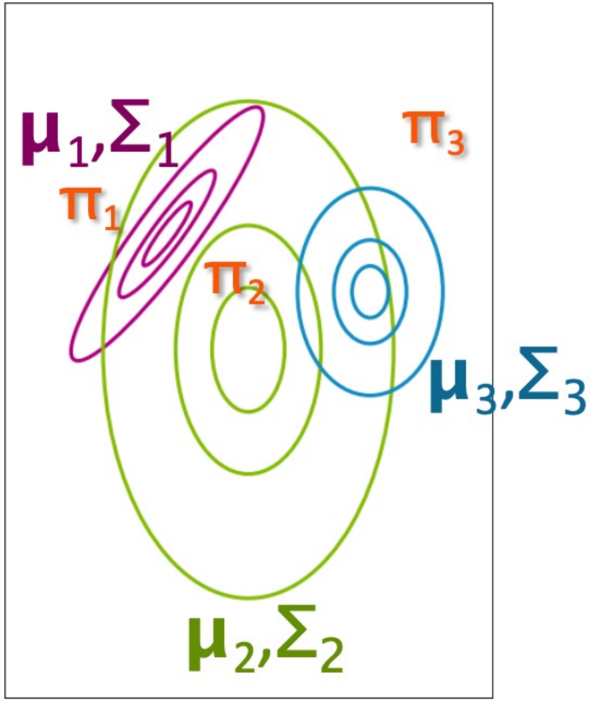
Mixture of Gaussians (1-D)

Each mixture component represents a unique cluster specified by a mean $\mu^{(j)}$ and variance $\sigma^{(j)}$



Mixture of Gaussians (multi-dimensional) (optional)

Each mixture component represents a unique cluster specified by a mean $\mu^{(j)}$ and covariance $\Sigma^{(j)}$



Expectation-Maximization Algorithm

Uses the MLE to maximize the likelihood that all datapoints get assigned to the given Gaussian distributions.

Chicken and egg problem

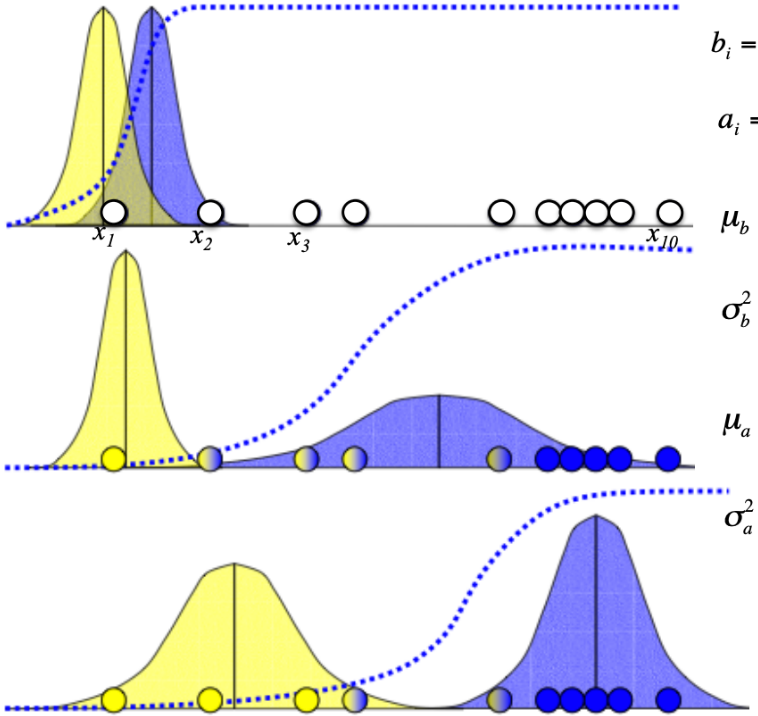
- Need to know the means and covariances of clusters to categorize the points
- Need to know the points for each cluster to estimate the means and covariances

Algorithm

- Start with k randomly placed Gaussian means and covariances that represent k clusters
- Repeat until convergence:
  - For each point: Calculate the probability that each point belong to a certain cluster
  - Adjust the means and covariances based on the calculated probabilities

Example (1-D) (optional)

EM: 1-d example



$$P(x_i \mid b) = \frac{1}{\sqrt{2\pi\sigma_b^2}} \exp\left\{-\frac{(x_i - \mu_b)^2}{2\sigma_b^2}\right\}$$

$$b_i = P(b \mid x_i) = \frac{P(x_i \mid b)P(b)}{P(x_i \mid b)P(b) + P(x_i \mid a)P(a)}$$

$$a_i = P(a \mid x_i) = 1 - b_i$$

$$\mu_b = \frac{b_1 x_1 + b_2 x_2 + \ldots + b_n x_n}{b_1 + b_2 + \ldots + b_n}$$

$$\sigma_b^2 = \frac{b_1(x_1 - \mu_b)^2 + \ldots + b_n(x_n - \mu_b)^2}{b_1 + b_2 + \ldots + b_n}$$

$$\mu_a = \frac{a_1 x_1 + a_2 x_2 + \ldots + a_n x_n}{a_1 + a_2 + \ldots + a_n}$$

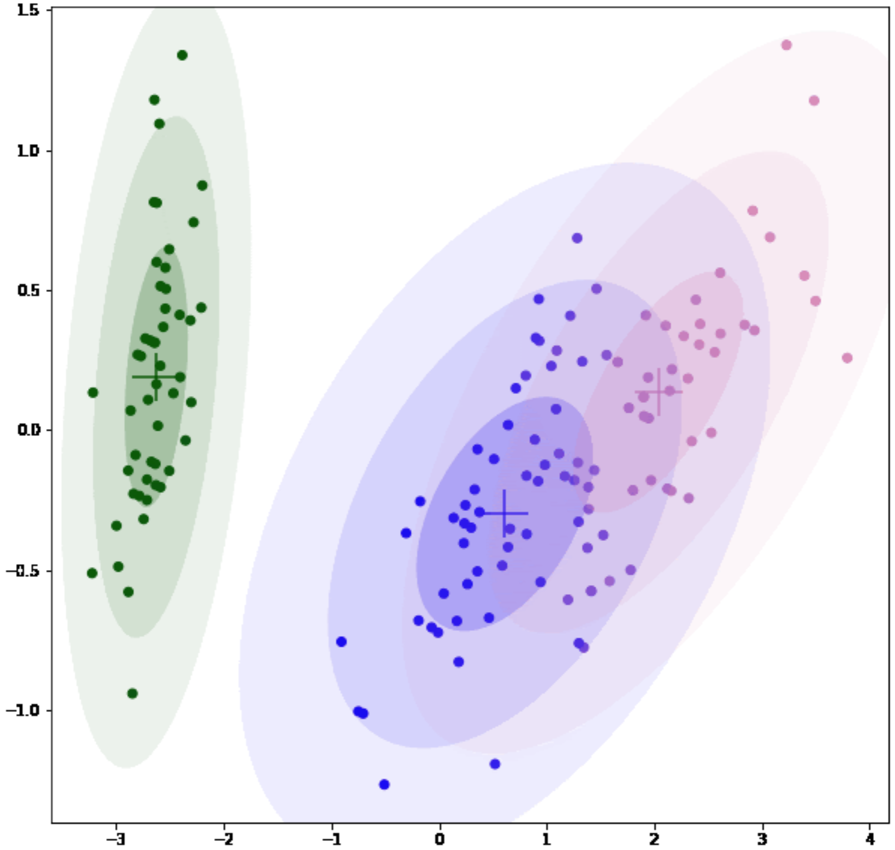
$$\sigma_a^2 = \frac{a_1(x_1 - \mu_a)^2 + \ldots + a_n(x_n - \mu_a)^2}{a_1 + a_2 + \ldots + a_n}$$

could also estimate priors:

$P(b) = (b_1 + b_2 + \ldots b_n) / n$

$P(a) = 1 - P(b)$

Visualization



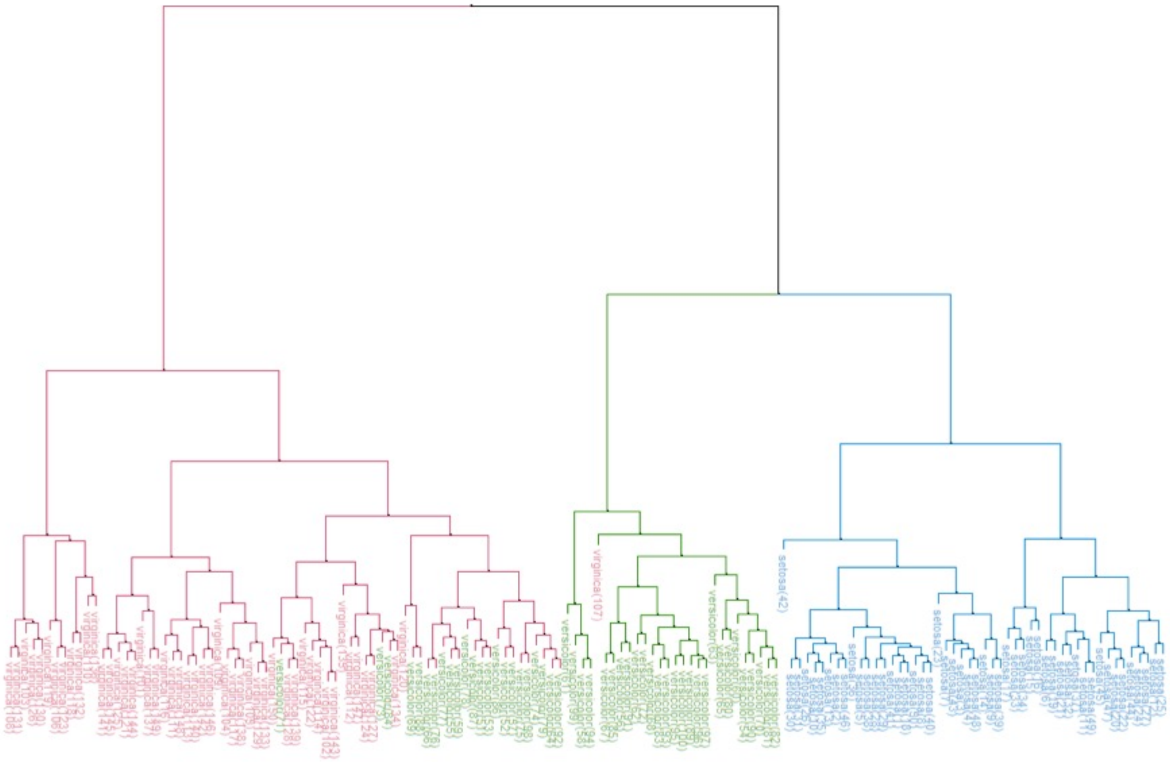
Expectation-Maximization Algorithm vs K-Means

K-Means is actually a **special case** of the EM algorithm, in that we let the probability of assigning a point to a cluster to be exactly 1, and for other clusters 0.

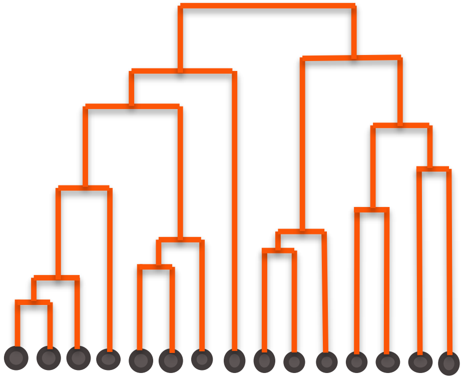
Converges to local minima like k-means

Hierarchical Clustering

Example: Species

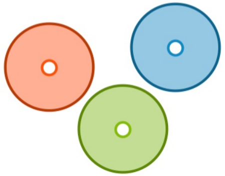


Motivation

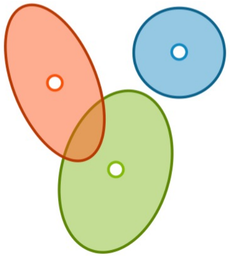


Finding Shapes

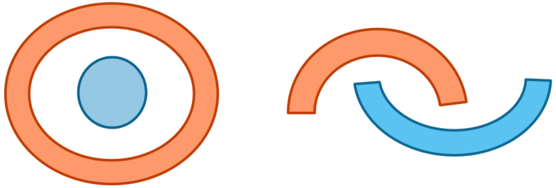
k-means



Mixture Models



Hierarchical Clustering



Types of Algorithms

Divisive, a.k.a. *top-down*

Start with all the data in one big cluster and then recursively split the data into smaller clusters

- Example: **recursive k-means**

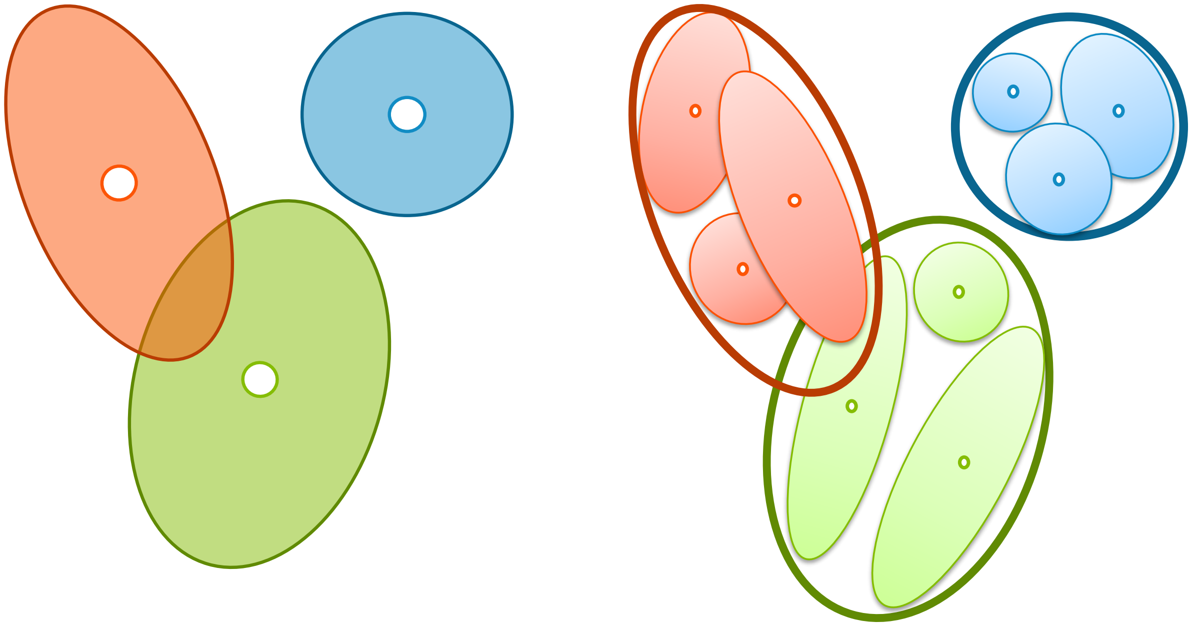
Agglomerative, a.k.a. *bottom-up*:

Start with each data point in its own cluster. Merge clusters until all points are in one big cluster.

- Example: **single linkage**

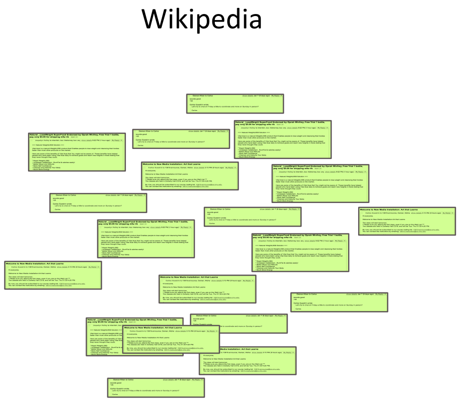
Divisive Clustering

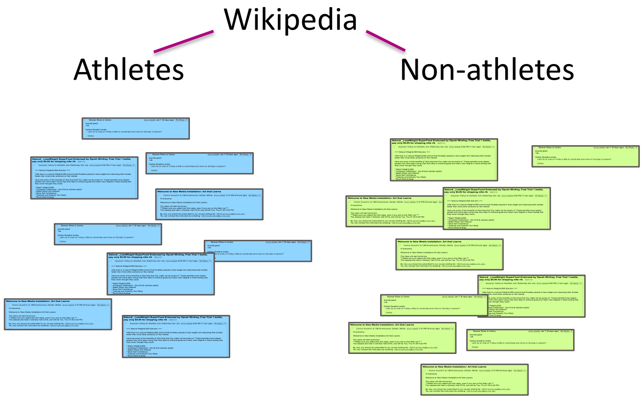
Start with all the data in one cluster, and then run k-means to divide the data into smaller clusters. Repeatedly run k-means on each cluster to make sub-clusters.

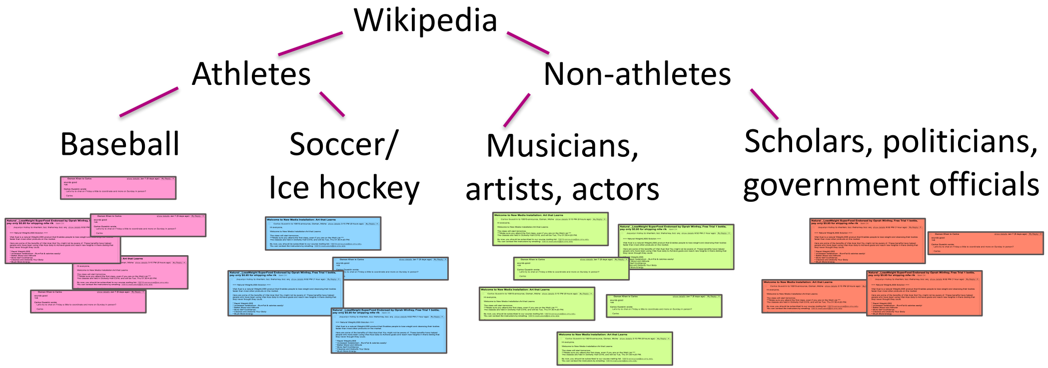


Example

Using Wikipedia







Choices to Make

For decisive clustering, you need to make the following choices:

Which algorithm to use

How many clusters per split

When to split vs when to stop

- **Max cluster size**
  Number of points in cluster falls below threshold
- **Max cluster radius**
  distance to furthest point falls below threshold
- **Specified # of clusters**
  split until pre-specified # of clusters is reached

Agglomerative Clustering

Algorithm at a glance

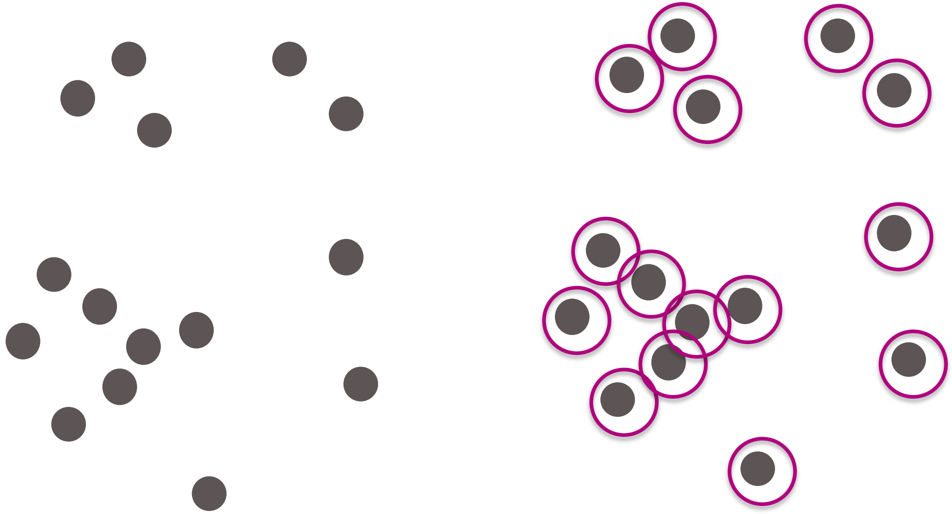
1. Initialize each point in its own cluster
2. Define a distance metric between clusters

While there is more than one cluster

3. Merge the two closest clusters

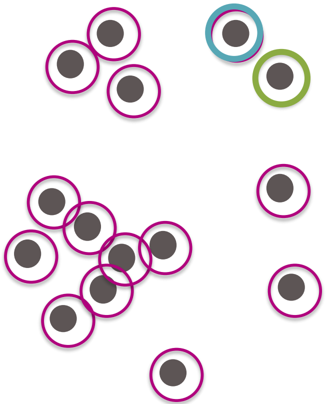
Step 1

1. Initialize each point to be its own cluster



Step 2

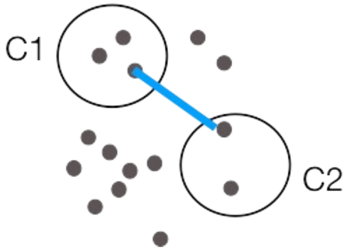
2. Define a distance metric between clusters



Single Linkage

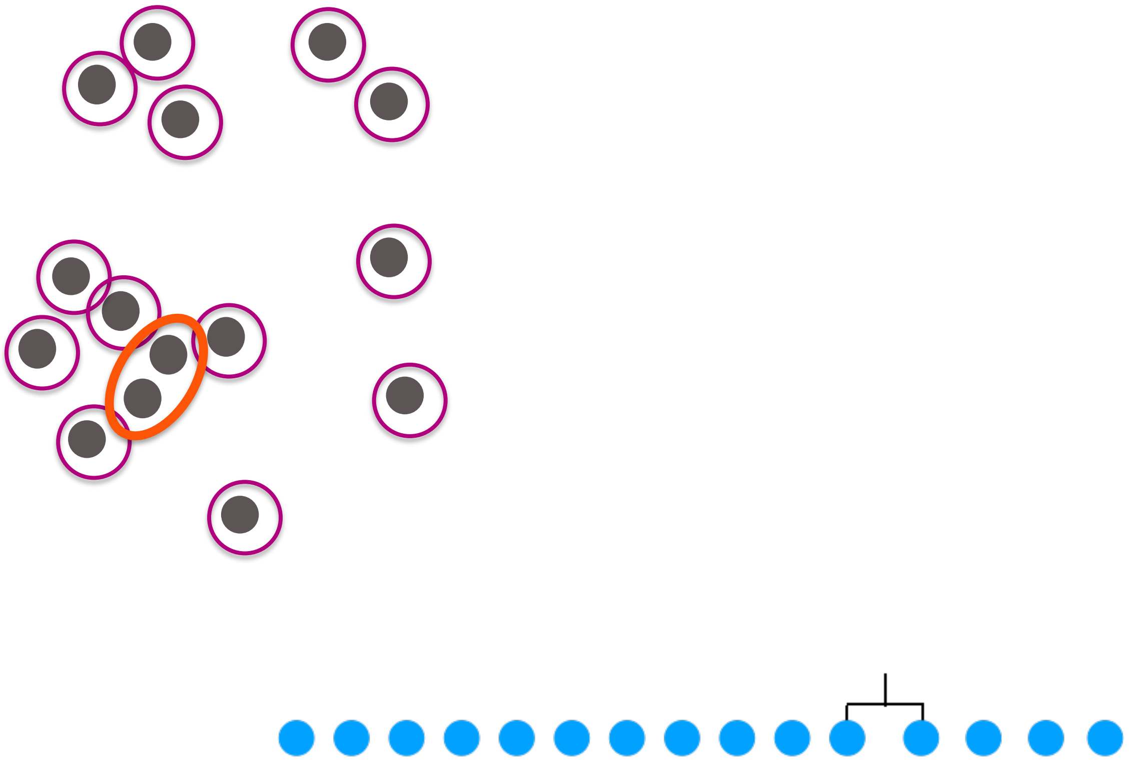
$$distance\left(C^{(1)}, C^{(2)}\right) = \min_{x^{(i)} \in C^{(1)}, x^{(j)} \in C^{(2)}} d\left(x^{(i)}, x^{(j)}\right)$$

This formula means we are defining the distance between two clusters as the smallest distance between any pair of points between the clusters.

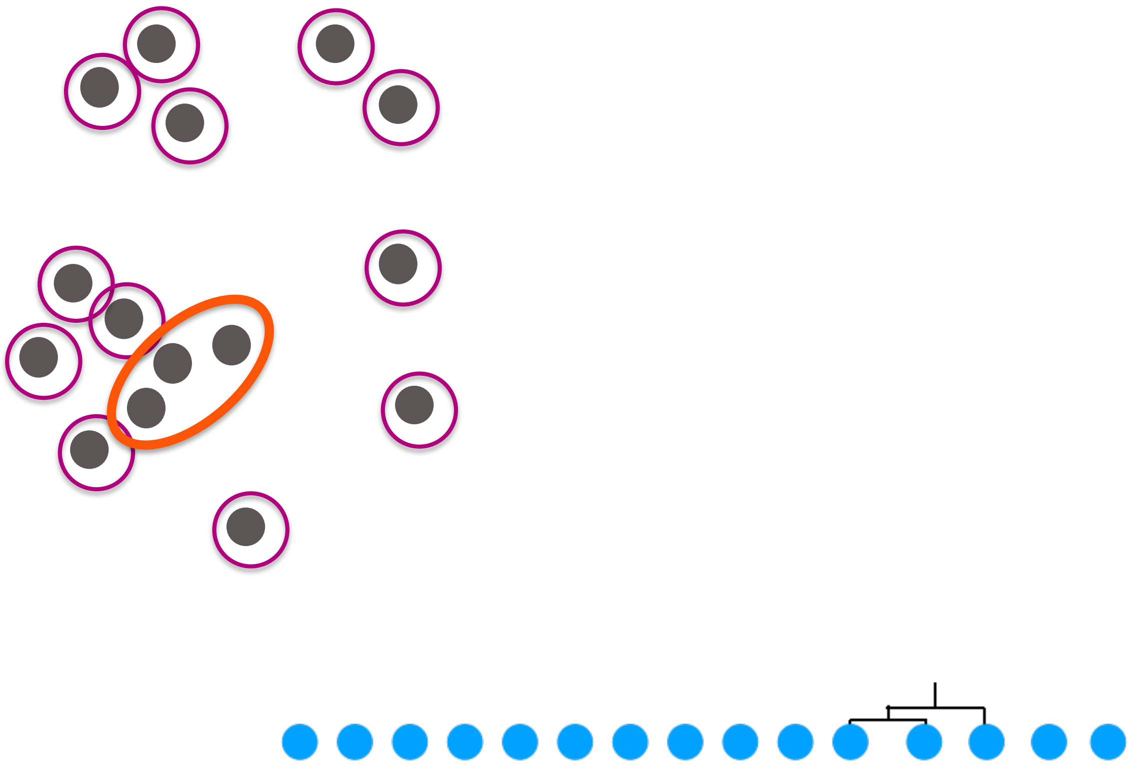


Step 3

Merge closest pair of clusters

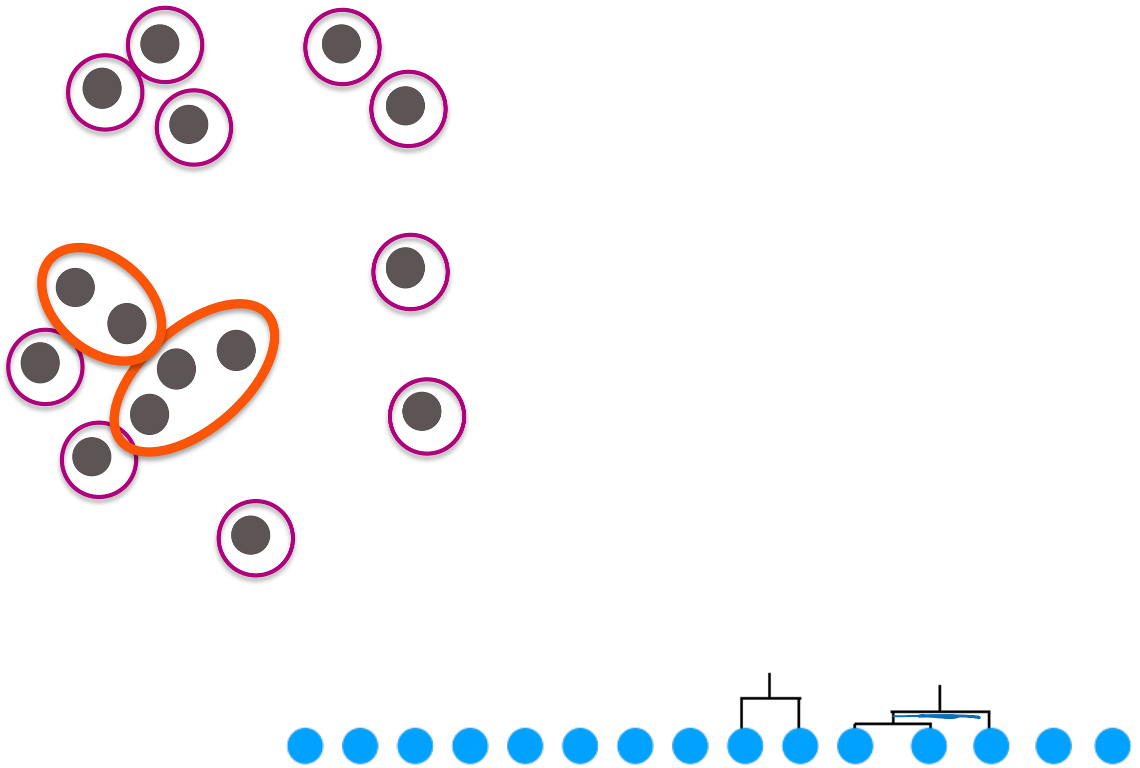


Repeat

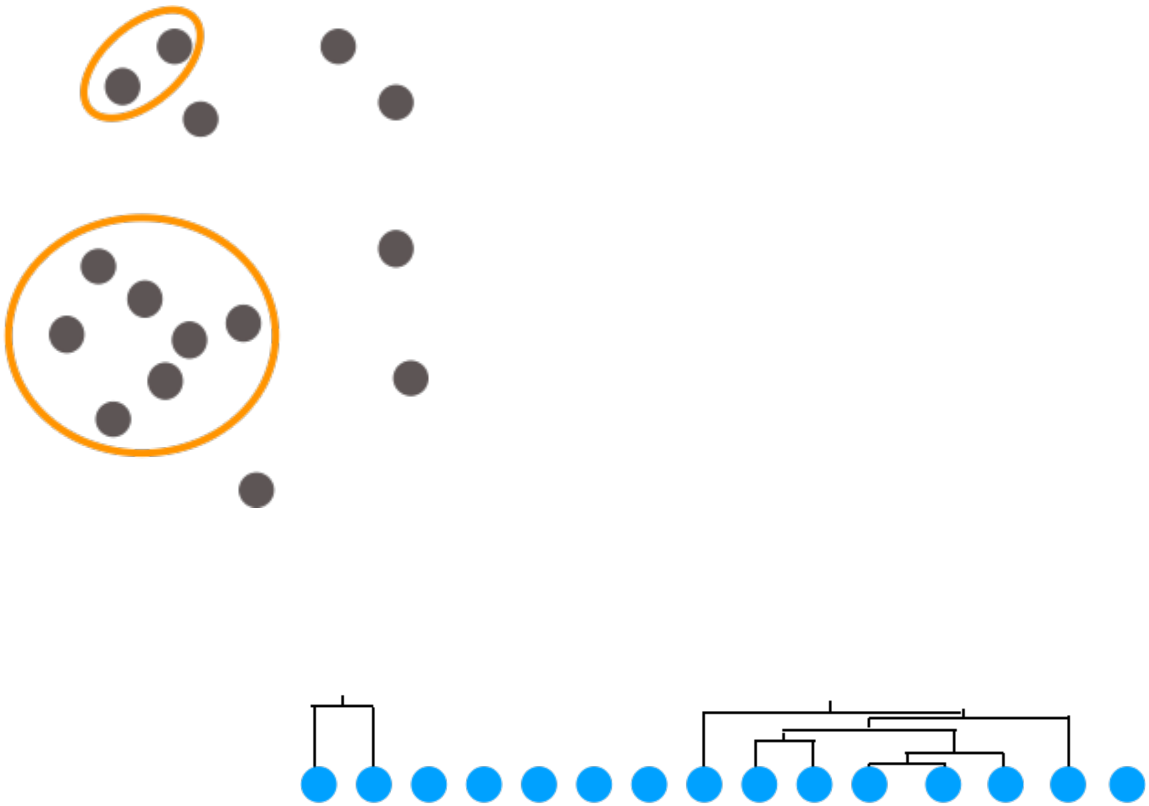


Repeat

Notice that the height of the dendrogram is growing as we group points farther from each other

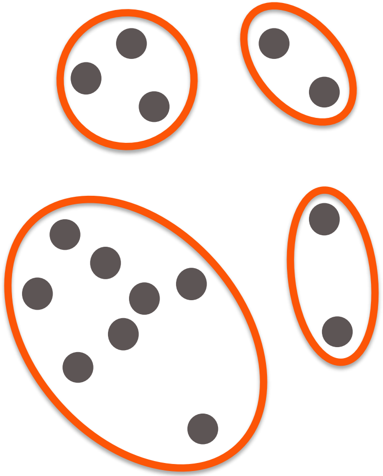


Repeat

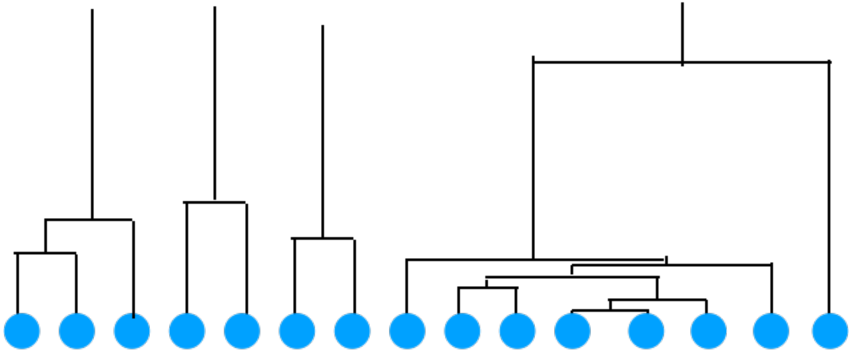


Repeat

Looking at the dendrogram, we can see there is a bit of an outlier!

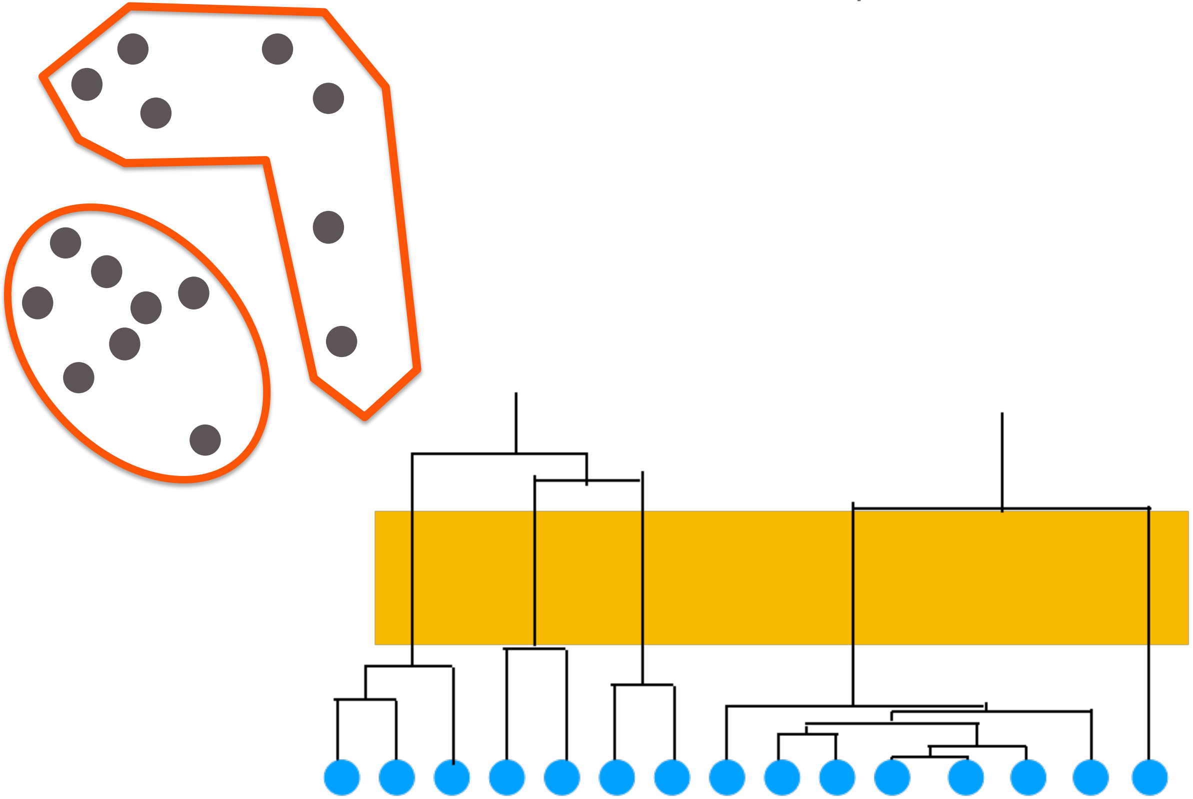


Can tell by seeing a point join a cluster with a really large distance.



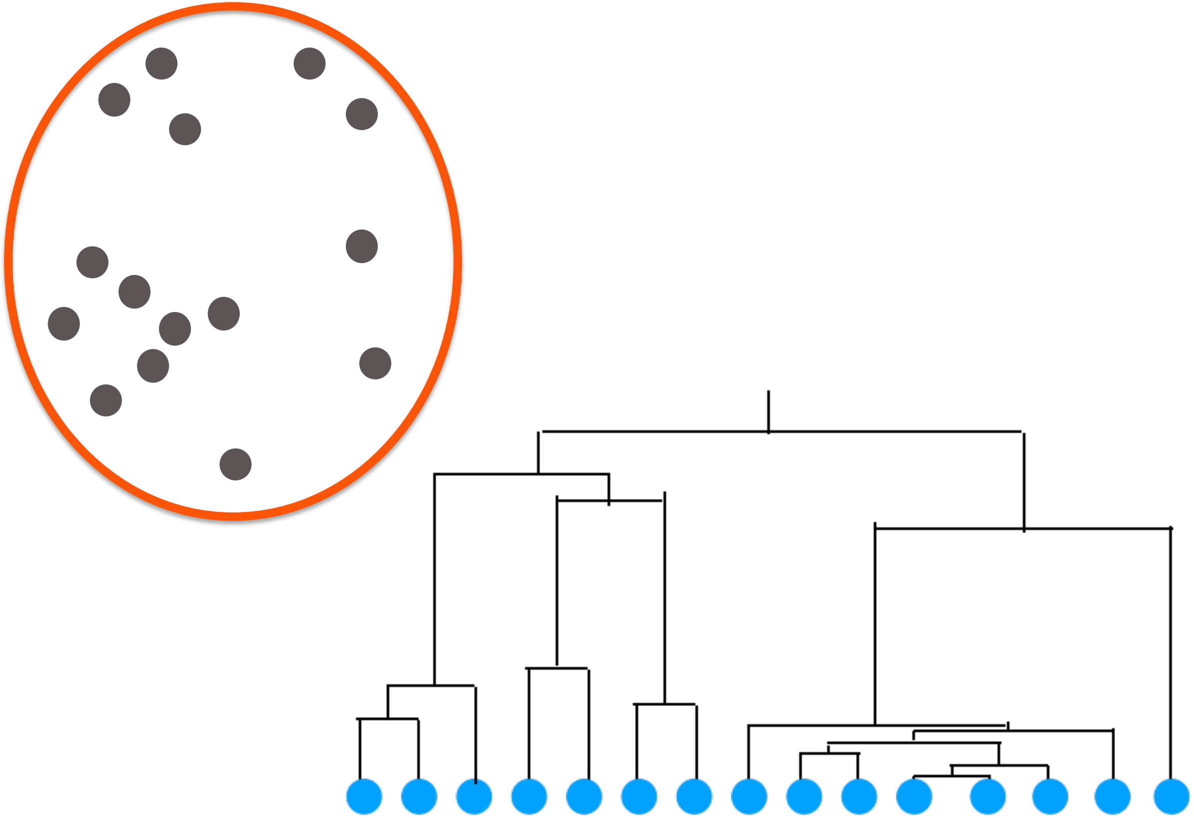
Repeat

The tall links in the dendrogram show us we are merging clusters that are far away from each other

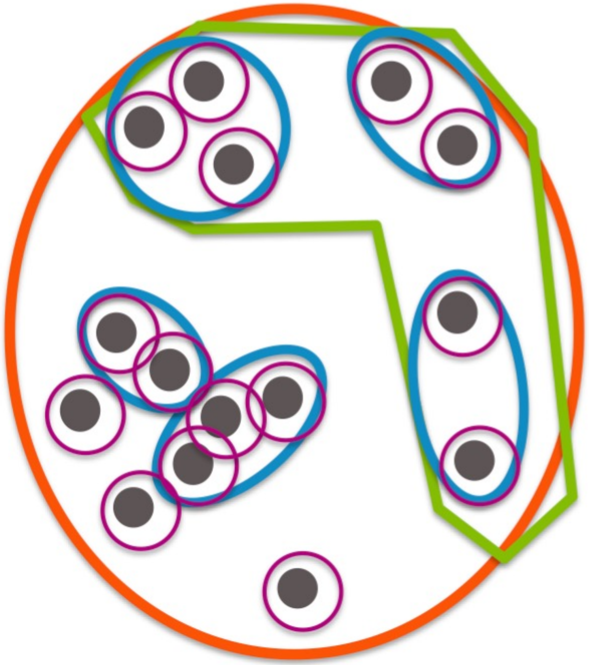


Repeat

Final result after merging all clusters



Final Result

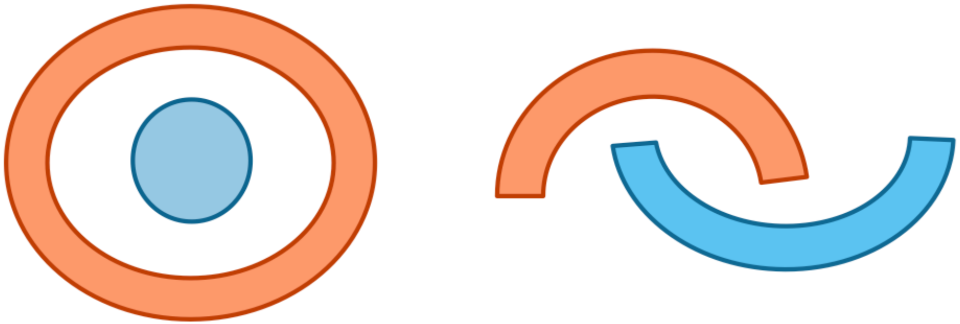


Brain Break

3:10

Agglomerative Clustering

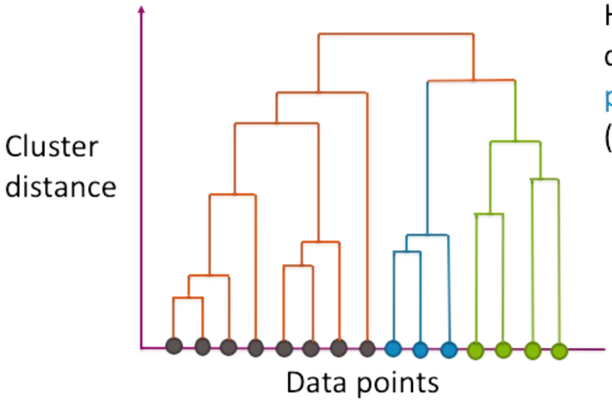
With agglomerative clustering, we are now very able to learn weirder clusterings like



Dendrogram

x-axis shows the datapoints (arranged in a very particular order)

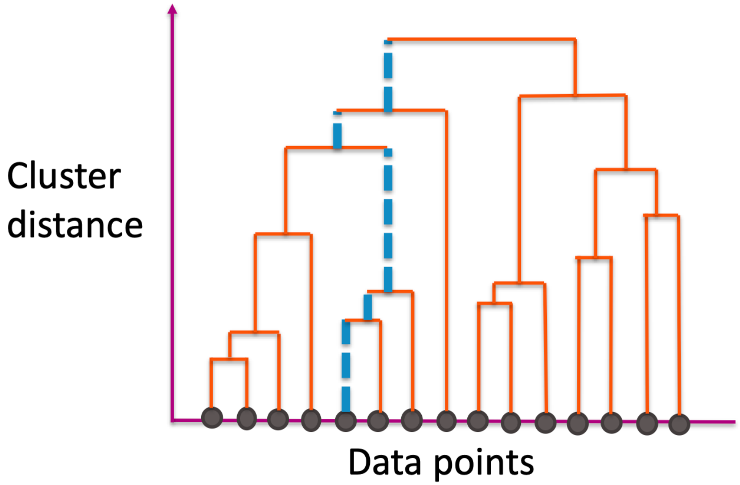
y-axis shows distance between pairs of clusters



Height here indicates min distance between blue pts and green pts (2 clusters)

Dendrogram

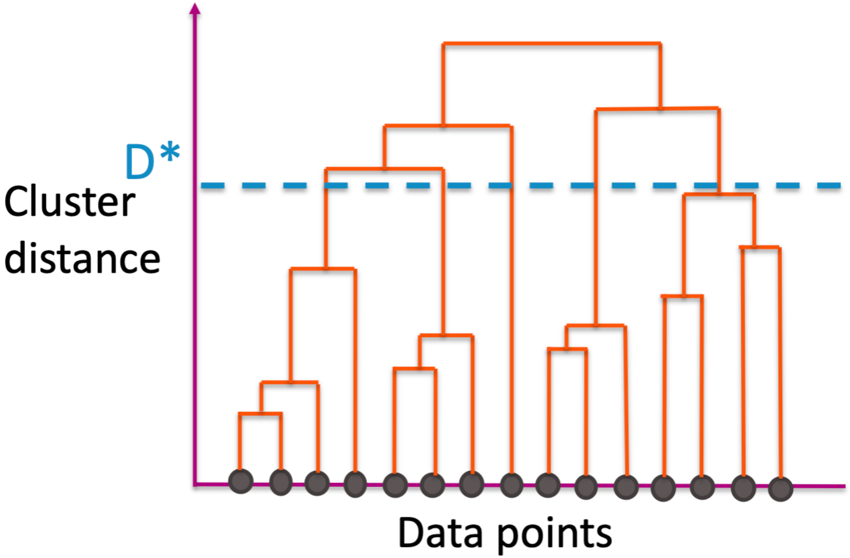
The path shows you all clusters that a single point belongs and the order in which its clusters merged



Cut Dendrogram

Choose a distance D^* to "cut" the dendrogram

- Use the largest clusters with distance < D^*
- Usually ignore the idea of the nested clusters after cutting



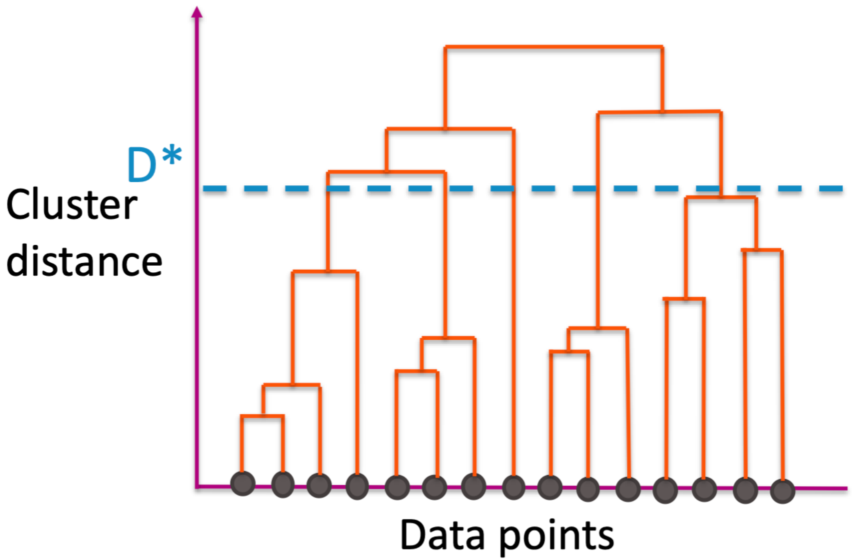
Poll Everywhere

Think

1 min

pollev.com/cs416

How many clusters would be have if we use this threshold?



1:00

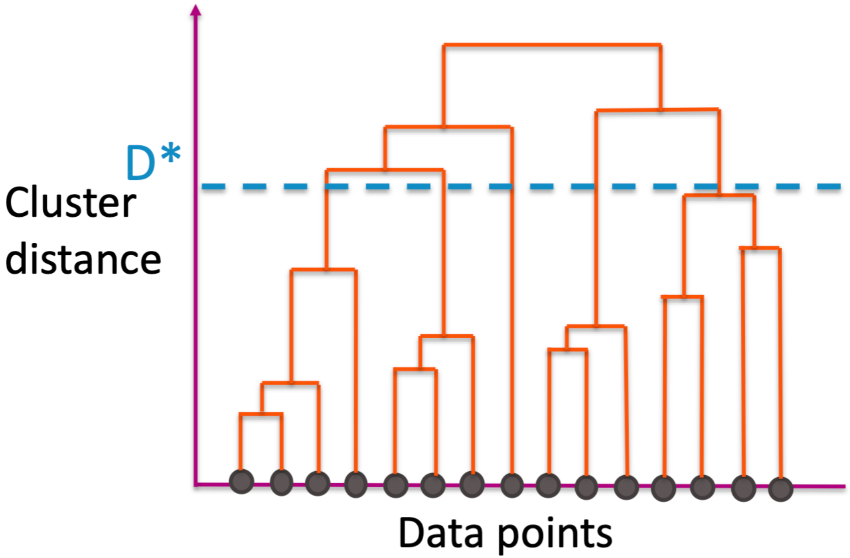
Poll Everywhere

Think

2 min

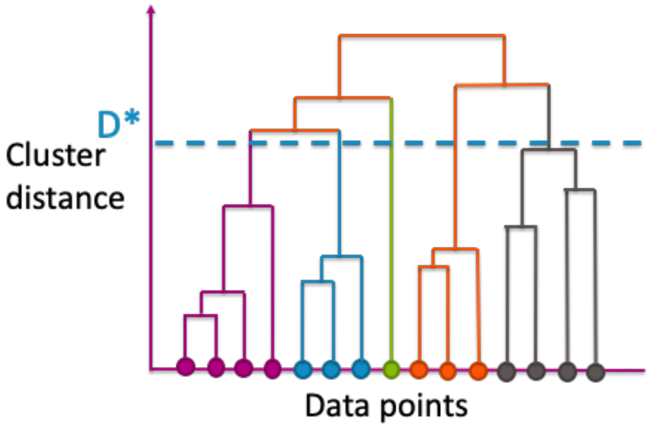
pollev.com/cs416

How many clusters would we have if we use this threshold?



Cut Dendrogram

Every branch that crosses D^* becomes its own cluster



Choices to Make

For agglomerative clustering, you need to make the following choices:

Distance metric $d(x_i, x_j)$

Linkage function

- Single Linkage:

$$\min_{x_i \in C_1, x_j \in C_2} d(x_i, x_j)$$

- Complete Linkage:

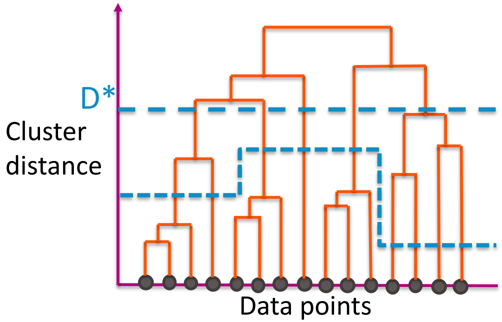
$$\max_{x_i \in C_1, x_j \in C_2} d(x_i, x_j)$$

- Centroid Linkage

$$d(\mu_1, \mu_2)$$

- Others

Where and how to cut dendrogram



Practical Notes

For visualization, generally a smaller # of clusters is better

For tasks like outlier detection, cut based on:

- Distance threshold
- Or some other metric that tries to measure how big the distance increased after a merge

No matter what metric or what threshold you use, no method is "incorrect". Some are just more useful than others.

Computational Cost of Agglomerative Hierarchical Clustering

Computing all pairs of distances is pretty expensive!

A simple implementation takes $\mathcal{O}(n^2 \log(n))$

Can be much implemented more cleverly by taking advantage of the **triangle inequality**

"Any side of a triangle must be less than the sum of its sides"

Best known algorithm is $\mathcal{O}(n^2)$

Agglomerative *vs* Divisive

Divisive clustering is more complicated to implement than agglomerative clustering, since we have to specify different values of k in different recursive loops.

Agglomerative clustering makes decisions by considering the local patterns or neighbor points without initially considering the global distribution of data. These early decisions cannot be undone. On the other hand, divisive clustering considers the global distribution of data when making top-level partitioning decisions.

Differences between k-means clustering and agglomerative hierarchical clustering

Hierarchical clustering can't handle big data well but k-means can. This is because the time complexity of K Means is mostly linear i.e. $O(n \ * \ k \ * I)$ while that of hierarchical clustering is quadratic i.e $O(n^2)$ (I is the number of iterations)

In K-means clustering, since we start with random choice of clusters, the results produced by running the algorithm multiple times might differ. While results are reproducible in agglomerative hierarchical clustering

K-means is found to only work well when each cluster is hyper spherical (like circle in 2D, sphere in 3D), but

K-means clustering requires prior knowledge. For hierarchical clustering, you can stop at whatever number of clusters you find appropriate in hierarchical clustering by interpreting the dendrogram.

Missing Data

Missing Data

Data in the real-world is rarely clean or as nicely structured as data provided to you on a HW in class. You saw this in HW6!

One common way data can be messy (but not the only one!) is to have missing values.

> This usually takes the form of a NaN or some special value (e.g., an empty string or -1).

Just like how there isn't ever one right answer for modeling, how you deal with missing data will not have one right answer either!

> Usually depends on domain experience!

Missing Data

Missing data can happen at either

- Training time
- Prediction time (e.g., testing or after deploying)

Credit	Term	Income	y
excellent	3 yrs	high	safe
fair	?	low	risky
fair	3 yrs	high	safe
poor	5 yrs	high	risky
excellent	3 yrs	low	risky
fair	5 yrs	high	safe
poor	?	high	risky
poor	5 yrs	low	safe
fair	?	high	safe

Loan application may be
3 or 5 years

Strategy 1: Skipping

The simplest strategy is just completely ignore missing values so you don't have to deal with them.

This can take the form of

- Dropping rows with missing values
- Dropping features (columns) with missing values

Which to drop depends on how much data is missing / how important those entities are:

- If only one training row has a missing value, dropping it doesn't seem to bad
- If only a few features (out of many) have missing values, maybe just drop those!

Strategy 1: Skipping (Pros/Cons)

Pros

- Very easy to understand/explain
- Can be applied to any model

Cons

- Might be removing useful information
- When is it better to remove examples vs. features?
- Doesn't help if data is missing at prediction time

Strategy 2: Sentinel Values

Idea: Replace missing data with some default value

Credit	Term	Income	y
excellent	3 yrs	high	safe
fair	UNK	low	risky
fair	3 yrs	high	safe
poor	5 yrs	high	risky
excellent	3 yrs	low	risky
fair	5 yrs	high	safe
poor	UNK	high	risky
poor	5 yrs	low	safe
fair	UNK	high	safe

Strategy 2: Sentinel Values (Pros/Cons)

Pros

Fairly simple to describe

Efficient fix and works at prediction time as well

Works well for categorical features (treats missingness as an important value in its own).

Cons

Only works well for features that are already categorical. Numeric features have no clear sentinel value.

Strategy 3: Imputation

Use some heuristic (or learning) to fill in missing data with better guesses for their values.

A simple approach:

Categorical features: Use most popular (mode) of non-missing values

Numeric features: Use mean or median of non-missing values

Complex approach:

Use a learning algorithm to learn relationships between the other features and the features with missing values. Fill in missing values with some learned model.

- Many algorithms use a back-and-forth processes like EM used in clustering!

Strategy 3: Imputation (Pros/Cons)

Pros

- Usually easy to understand and implement (if using simple approach)
- Can be applied to any model
- Can be used at prediction time: Use same imputation rules

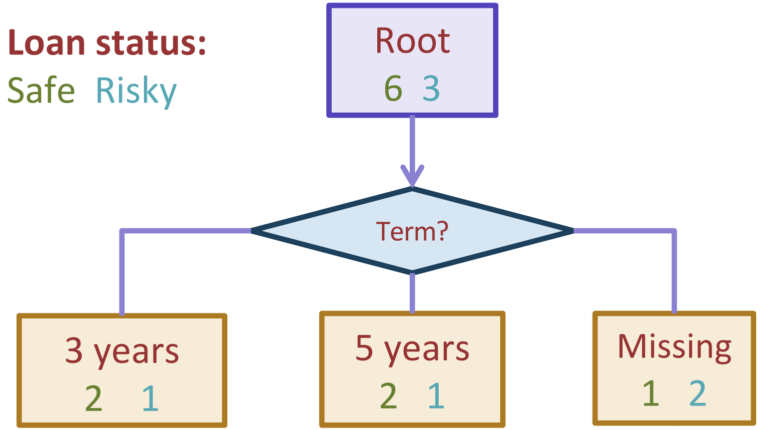
Cons

- May result in systematic errors (ask: why are certain values missing)
- Missing values could signal of their own and this removes them (e.g., credit-card fraud)

Strategy 4: Modify Algorithm

Use a new type of model that is robust to the presence of missing values.

For example, implement a new decision tree from scratch that can handle missing values (e.g., makes a new branch if a value is missing)



Strategy 4: Modify Algorithm (Pros/Cons)

Pros

Very similar to sentinel values in terms of pros but can also handle numeric features

Generally can have more accurate predictions

Cons

Requires implementing a new type of model

- Maybe easy for decision trees, but other types???

Miss Data - Recap

There are a lot of approaches to handling missing values. Note that missing values is a common source of messy data, is just one out of an infinite number of ways your data could be difficult to work with.

There will never be "one right strategy", how you handle missing data is a modeling choice just like every other modeling you choice you make.

Concept Inventory

This week we want to practice recalling vocabulary. Spend 10 minutes trying to write down all the terms for concepts we have learned in this class and try to bucket them into the following categories.

Regression

Classification

Document Retrieval

Misc – For things that fit in multiple places or none of the above

You don't need to define/explain the terms for this exercise, but you should know what they are!

Try to do this for at least 5 minutes from recall before looking at your notes!