CSCI 4022 Spring 2021
More Clustering: GMMs and K-means

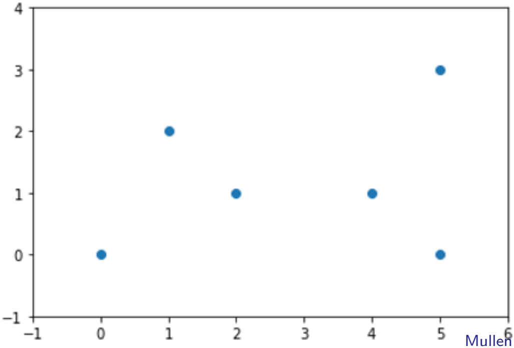
1st of 1000 example correlations:

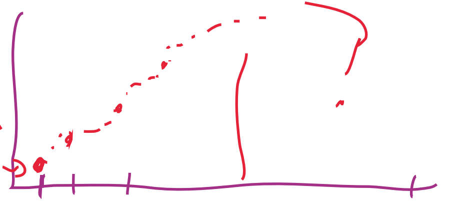
reasonable values

"rejection" region

extreme rare values

Example: Consider the data set $(0,0), (1,2), (2,1), (4,1), (5,0), (5,3)$. Use hierarchical clustering and Euclidean distance to group the data into 2 clusters. If there are ties in distance, merge first the data points with lower x-coordinates.





$[Z(\text{point1}) - Z(\text{point2})]^2$ for each pt in that bin.

plt.plot (20 bins, 20 avg. vals)

Announcements and To-Dos

Announcements:

1. HW 1 due tonight! 3p-6p Zach OH.
2. HW 2 posted tomorrow probably (2 problems; a quick theory one and a larger shingling/minhash exercise)

Weekly min form comments: - time & memory savings

minhasing (various parts; min column, big picture, etc.)

example of hierarchical

homework stuff from intro course

Clustering Recap

Examples: Data points could represent...

Clustering Applications

- Different characteristics of songs:
  Goal: cluster together similar songs into genres
- Vehicle weights, milages, other characteristics:
  Goal: cluster together similar vehicles into classes (SUV, sedan, hybrid...)
- Sky object radiation intensities into frequency ranges
  Goal: cluster together into groups of similar objects.
- Words in a document
  Goal: cluster together into groups of similar topics.

Clustering Issues

- Clustering looks easy in two dimensions
- Clustering small amounts of data looks easy
- in most cases, looks are not deceiving...
- but many applications have not 2, but 10 or 10,000 dimensions. What does that even *look* like?
- High-dimensional spaces look different: almost all pairs of points are at about the same distance!

Hierarchical Implementation

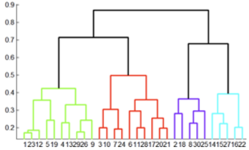
Implementation Notes

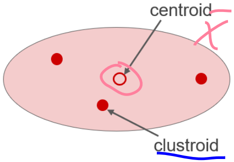
At each step, we compute *all* distances between pairs of clusters. Then merge the nearest two clusters.

Once a cluster is formed, it is represented only by its **centroid** (average) or **clustroid** (median/representative point)

With N data points, this is N^2 comparisons to make! (or $\binom{N}{2}$, at least).

IF we want k clusters at the end, and $k << N$, then we need to iterate about N times to merge down to k clusters. This means $\mathcal{O}(N^3)$ complexity, although the # of pairwise comparisons does shrink as points collapse into clusters.



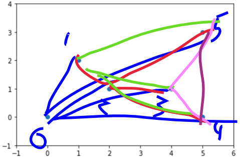


X average

$(\bar{X}, \bar{Y})$

Hierarchical Example

Example: Consider the data set $(0,0),(1,2),(2,1),(4,1),(5,0),(5,3)$. Use hierarchical clustering and Euclidean distance to group the data into 2 clusters. If there are ties in distance, merge first the data points with lower x-coordinates.

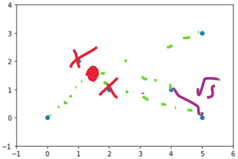


Hierarchical Example

Example: Consider the data set $(0,0),(1,2),(2,1),(4,1),(5,0),(5,3)$. Use hierarchical clustering and Euclidean distance to group the data into 2 clusters. If there are ties in distance, merge first the data points with lower x-coordinates.

Solution: We might initially construct a full distance matrix!

$$\begin{array}{c|cccccc} & (0,0) & (1,2) & (2,1) & (4,1) & (5,0) & (5,3) \\ \hline (0,0) & & {\scriptstyle 2}\,\sqrt{5} & \sqrt{5} & \sqrt{17} & 5 & \sqrt{34} \\ (1,2) & & & \sqrt{2} & \sqrt{10} & \sqrt{20} & \sqrt{17} \\ (2,1) & & & & 2 & \sqrt{10} & \sqrt{13} \\ (4,1) & & & & & \sqrt{2} & \sqrt{5} \\ (5,0) & & & & & & 3 \\ (5,3) & & & & & & \end{array}$$

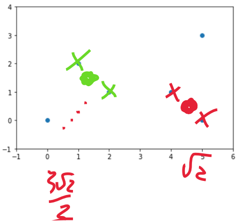


There are a couple of $\sqrt{2}$ in there, so we pick the tiebreaker of the lowest x-values, which are $(1,2)$ and $(2,1)$.

Hierarchical Example

Example: Consider the data set $(0, 0), (1, 2), (2, 1), (4, 1), (5, 0), (5, 3)$. Use hierarchical clustering and Euclidean distance to group the data into 2 clusters. If there are ties in distance, merge first the data points with lower x-coordinates.

Solution: Now we'd have a **cluster** inside our distance matrix. Points $(1, 2)$ and $(2, 1)$ get folded into a **cluster** with **centroid** at $(3/2, 3/2)$.
We could ostensibly recreate the matrix, but now in a 5x5 instead of 6x6 format. For this smaller problem, let's proceed visually, instead.



Hierarchical Example

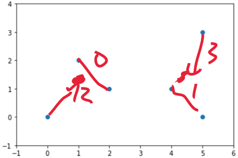
Step1: find min distance

cluster { Step 1a: combine each cluster in that min
Step 2: update centroid

Example: Consider the data set $(0,0),(1,2),(2,1),(4,1),(5,0),(5,3)$. Use hierarchical clustering and Euclidean distance to group the data into 2 clusters. If there are ties in distance, merge first the data points with lower x-coordinates.

Solution: full combine order

1. Combine $(1,2)$ and $(2,1)$ into group red
2. Combine $(4,1)$ and $(5,0)$ into group blue
3. Fold $(0,0)$ into red group.
4. Fold $(5,3)$ into blue group.
5. STOP: we're down to 2 groups, since every points is either red or blue.



$X=(x_1,x_2,x_3,x_4)$ $y=(y_1,y_2,y_3,y_4)$ dist(x,y)

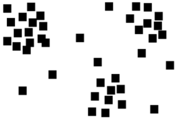
k-means

GOAL: find best "k" clusters (# of groups = k).

We can save considerable amounts of time by instead using the *k-means* algorithm.

Setup for k-means:

1. Requires specification of a norm or distance measure: typically an L-norm or Euclidean distance.
2. Fix a value for k, the total number of clusters.
3. Initialize the clusters in some fashion, typically with *only one point per cluster*. Some options: Pick k!
   - 3.1 Random plan: pick k points totally at random to each be in different clusters (slow for large data, unstable).
   - 3.2 Hierarchical plan: do hierarchical clustering to get to k clusters from a (small) *subset* of the data, then randomly select one point from each cluster
   - 3.3 Pick a first point randomly, then subsequently pick subsequent points to be *as far as possible* from each of the previous points. (i.e. append point with maximal minimum distance to the set of chosen points)

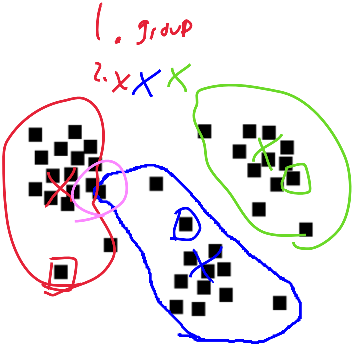


(Pick a subset $n<200$)

k-means

Iteration Scheme for k-means:

1. *For each point*
    1.1 Assign that point to the cluster whose centroid it is closest to.
2. Then, *For each cluster*
    2.1 Update the centroid of that cluster to reflect any added or lost points.
3. Repeat until **convergence**.
    3.1 Points may stop moving at all between clusters...
    3.2 or centroids will stabilize and not move (much)

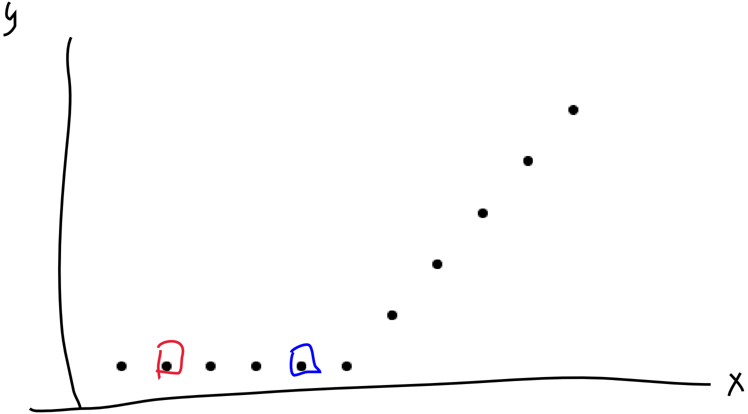


now closer to red

k-means Example

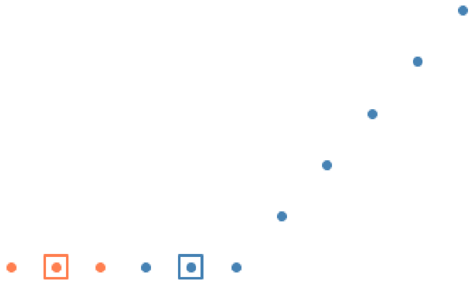
Suppose we have some 2-D data that mostly lies along a couple of lines.

Suppose that we “randomly” choose the 2nd and 5th points to initialize our clusters.



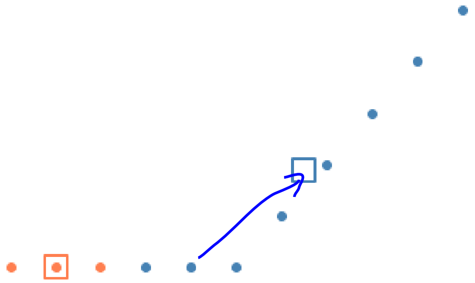
k-means Example

Step 1: assign points to clusters



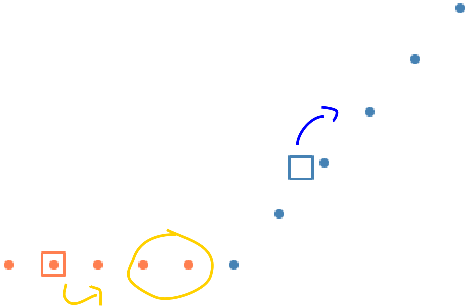
k-means Example

Step 2: update cluster centroids



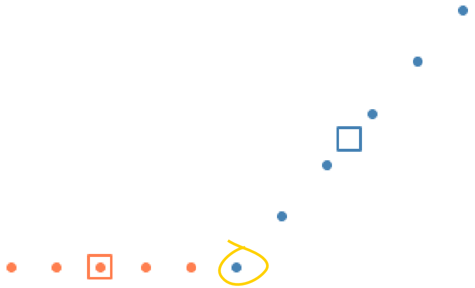
k-means Example

Step 1: assign points to clusters



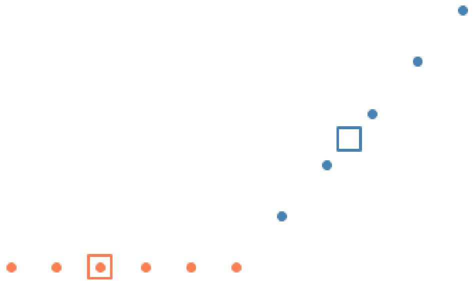
k-means Example

Step 2: update cluster centroids



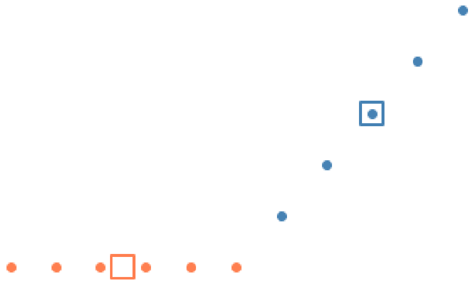
k-means Example

Step 1: assign points to clusters



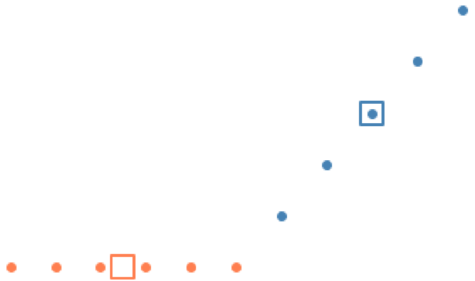
k-means Example

Step 2: update cluster centroids



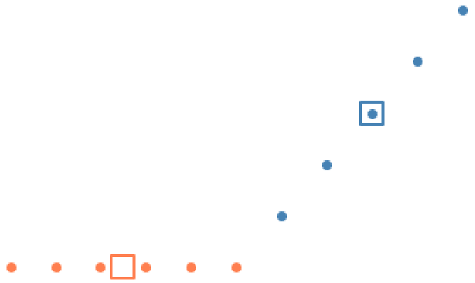
k-means Example

Step 1: assign points to clusters



k-means Example

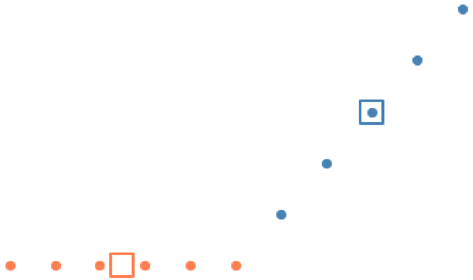
Step 2: update cluster centroids



k-means Example

Step 3: **BREAK**

Nothing has changed! Whether our convergence check was during step 1 or step 2, we'll break



k-means and k

check dist: n points → Kn complexity
k clusters

x →

x-s

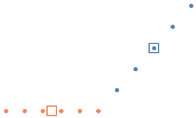
In this example, we choose $k = 2$ without putting any thought into it. But how do choose the correct value of k?

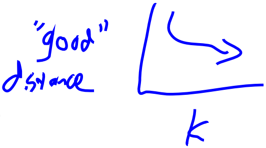
1. Sometimes an educated guess will work if you can visualize the data (at most 3 features/columns).

2. In higher dimensions, try a few *different* k and look at measures of how grouped up points **within** each cluster are.

   - Common measure for this: look at the average distance between each data point and its cluster's centroid.

   - This average distance will always decrease as k increases, but it will start changing very little after the "right" k.



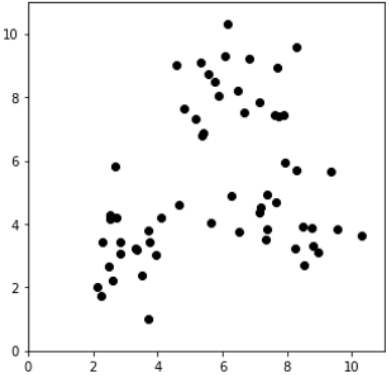


R^2 vs adjusted R^2

Choosing k

Approach: Try a few different k and look at the change in the average distance between each data point and its cluster's centroid. The average distance should decrease as k increases to about the right k, then change very little.

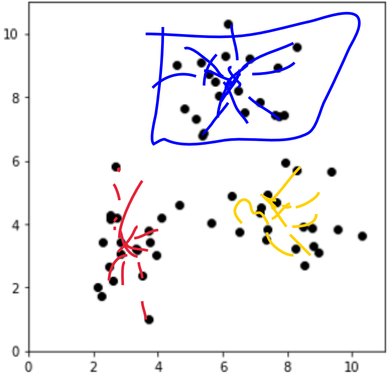
Example: What do you think? $k = 2?\, k = 3?,\ 4?\, 8?$



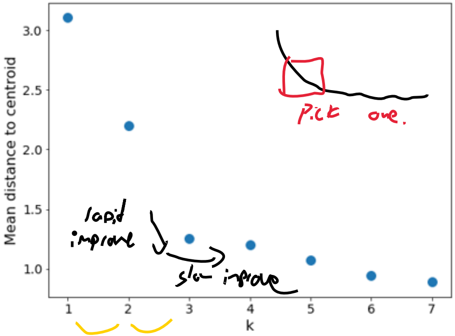
Choosing k

Approach: Try a few different k and look at the change in the average distance between each data point and its cluster's centroid. The average distance should decrease as k increases to about the right k, then change very little.

Example: What do you think? $k = 2$? $k = 3$?, 4? 8? Sanity check says: try $k = 3$

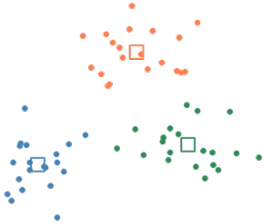


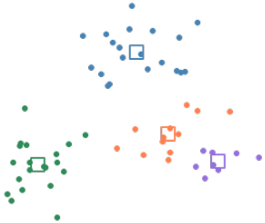
The **elbow plot** for k-means



Choosing k

Example: The *elbow plot* suggests that $k = 3$ was reasonable, as the curve levels off significantly there. Sometimes this curve is smoother and that's ok: pick a reasonable value or come up with a statistic for "best" k that penalizes extra terms.





Results for $k = 3$

Results for $k = 4$

Visually, notice that *not much benefit* from the $k = 3 \rightarrow k = 4$ transition. In practice, it mostly split the lower right grouping into 2! Since the other two clusters were unchanged, their mean-distance-to-centroid contributions didn't change either.

k-means and directions

k-means is a **circular** construction of clusters.

- Each cluster is uniquely defined by its *center*
- Points are assigned to clusters based on distance (or *radius* from center), without considering which *direction*
- This can be *very dangerous*. If there are linear trends in the data, points that are highly related can "look" far apart.

Units might also matter!

k-means and units

Units might also matter! Consider the data set with x_1: vehicle weight; x_2: vehicle mileage. These are on drastically different scales (lbs vs mpg?)

This could mean traditional distances fail entirely: what's the distance between $(1200lbs, 20mpg)$, $(1400lbs, 19mpg)$, $(1100lbs, 100mpg)$?

- Option A: *normalize* the data: replace each column with the original column, but subtract the mean then divide by the standard deviation.
- Option B: allow distances to somehow depend on *direction*.

We can let direction matter by fitting **ellipses** instead of circles!

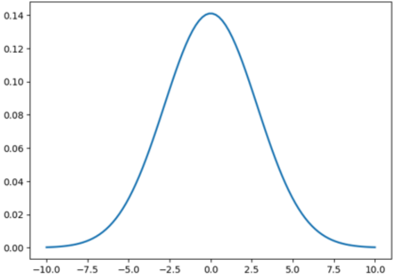
Our favorite ellipse is the **Normal Distribution**

The Gaussian (normal) distribution

You should recall the *normal distributionm Gaussian distribution*, or *bell curve*. In one dimension, it's a beautiful little function. (squared negative exponential).

The normal distribution has two arguments or *parameters*:

- μ: The mean or center of the curve. The most important *location*. A single scalar.
- σ^2: The variance/width of the curve. The most common measure of *dispersion* or spread. A single positive scalar.



A normal

The Gaussian (normal) distribution

Formalism

pdf: The *probability density function* of the normal distribution is

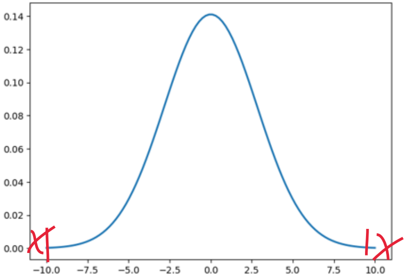
$\sim e^{-x^2}$

$$f(x;\mu,\sigma)=\frac{1}{\sqrt{2\pi\sigma^2}}e^{\frac{-(x-\mu)^2}{2\sigma^2}}$$

cdf: The **integral** of the pdf lets us answer probability questions like

$$P(a<X<b)=\int_a^b f(x)\,dx$$

- The normal is known for assigning *low probabilities* to outliers, or points *far from* μ



A normal

outliers

The 2-D Gaussian

In multiple dimensions, everything gets a bit more complex.

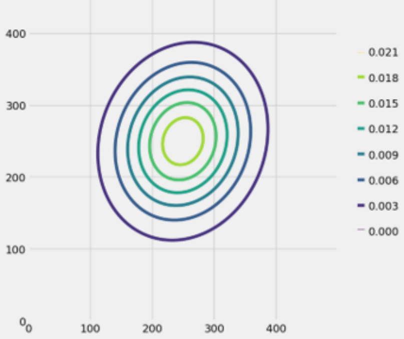
parameters:

μ: The mean or center of the surface. This is a 2-D (x, y) tuple, so we may write $\mu = (\mu_1, \mu_2) \in \mathbb{R}^2$

Σ: A 2×2 *covraiance* matrix whose entries are

$$\begin{bmatrix} \sigma_1^2 & cov(x_1, x_2) \\ cov(x_2, x_1) & \sigma_2^2 \end{bmatrix}$$

where σ_1^2 is the variance in the **first axis direction**, σ_2^2 is the variance in the **second axis direction**, and



A 2-D normal's contour plot

$$cov(x_1, x_2) = E\left[(X_1 - E[X_1])\,(X_2 - E[X_2])\right]$$

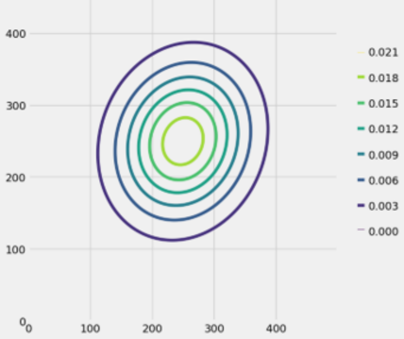
The 2-D Gaussian

This is actually **5** unique parameters.

parameters:

- μ_1: The location of the center in the axis-1 dimension.
- μ_2: The location of the center in the axis-2 dimension.
- σ_1^2: The variance in the axis-1 direction.
- σ_2^2: The variance in the axis-2 direction.
- cov: $cov(x_1, x_2)$The covariance of the data set (dimension one covariance with dimension 2).

Usage: $\text{NP.COV}(\text{X}1,\ \text{Y}1)$



A 2-D normal's contour plot

The 2-D Gaussian: variance and covariance

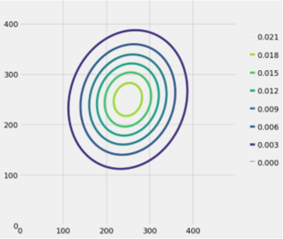
1. The off-diagonal arguments of Σ lead to *rotations.*

   Results: if $cov(x_1, x_2) = 0$, then the Gaussian is an ellipse that's oriented **vertically/horizontally** (i.e. the semi-major and semi-minor axis are parallel to the coordinate axes.).
2. σ_1^2 and σ_2^2 determine the widths in their respective coordinate directions

   Results: if $cov(x_1, x_2) = 0$ **and** $\sigma_1 = \sigma_2$, the contours would be circles... like k-means.

   Allowing σ_1 and σ_2 to vary stretches the circle in the corresponding direction. This gives a new method of clustering: **Gaussian Mixture Models**

$$\begin{bmatrix} \sigma_1^2 & cov(x_1, x_2) \\ cov(x_2, x_1) & \sigma_2^2 \end{bmatrix}$$



The Gaussian Mixture Model (GMM): 1D Example

Motivating example/cautionary tale: Suppose you go to Chuck E. Cheese's for your niece's birthday party. As you look around, you start to feel rather self-conscious because there seem to be very few people around your age. Needing to pass the time, because you feel so, so awkward, you collect data on everyone's ages.

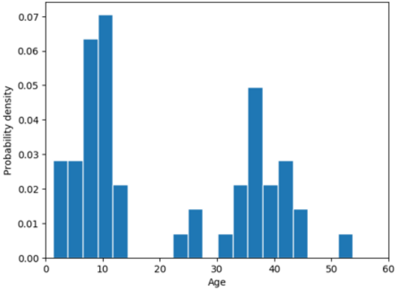
The Gaussian Mixture Model (GMM): 1D Example

Motivating example/cautionary tale: Suppose you go to Chuck E. Cheese's for your niece's birthday party. As you look around, you start to feel rather self-conscious because there seem to be very few people around your age. Needing to pass the time, because you feel so, so awkward, you collect data on everyone's ages.

You are then kicked out of and permanently banned form Chuck E. Cheese's because you were accosting children and asking about their ages...

... Fair enough.

Now that your afternoon is freed up, you plot up a histograme of the age data you so creepily and painstakingly collected. It looks like this:

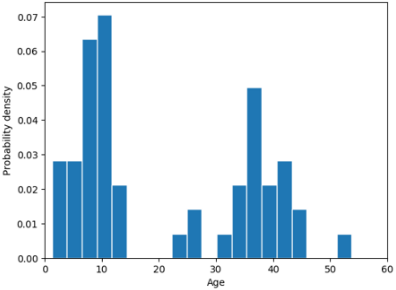


The Gaussian Mixture Model (GMM): 1D Example

The task: How can we best model this distribution?

It's clearly **bimodal**, so a simple normal Gaussian is not appropriate.
Instead, we view it as possibly the combination of **two** distributions:

1. The **kids'** ages seem like they might be reasonably Gaussian, aside from the pesky fact of non-negativity on ages.
2. The **parents'** ages might also be modeled by a *different* Gaussian distribution
3. This would make the overall distribution of patrons' ages a *Gaussian mixture model*



The Gaussian Mixture Model (GMM): 1D Example

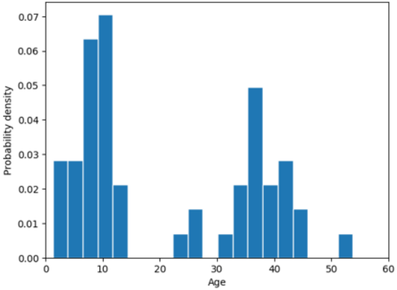
The whole model:
Kids' ages: a normal $X_1 \sim N(\mu_1, \sigma_1^2)$

Parents' ages: a normal $X_2 \sim N(\mu_2, \sigma_2^2)$

Who's who: a patron could be either of these at some *mixing proportion*. Define the *Bernoulli* random variable Δ where the probability that a patron is an adult $(\Delta = 1)$ by π. Then:

All patrons' ages:

$$X = (1 - \Delta) \cdot X_1 + \Delta \cdot X_2$$



The Gaussian Mixture Model (GMM): 1D Example

The whole model:

$$X = (1 - \Delta) \cdot X_1 + \Delta \cdot X_2$$

Let's unpack, since this is secretly just adding up all the possible ways we can observe a specific age:

$$\underbrace{X}_{\text{Prob of specific age}} = \overbrace{(1 - \Delta)}^{\text{Prob person is adult}} \cdot \underbrace{X_1}_{\text{Prob an ``adult'' is that age}} + \overbrace{\Delta}^{\text{Prob person is child}} \cdot \underbrace{X_2}_{\text{Prob a ``child'' is that age}}$$

The Gaussian Mixture Model (GMM): 1D Example

The whole model: $X = (1 - \Delta) \cdot X_1 + \Delta \cdot X_2$

Definition: The GMM is a *generative* model, since it specifies the probabilities for new data points.

1. Sample or simulate a Δ with a coin flip or NP.RANDOM.CHOICE
2. Based on Δ, sample a random normal from:
   - 2.1 X_1 as a $N(\mu_1, \sigma_1^2)$ if $\Delta = 0$ **OR**
   - 2.2 X_2 as a $N(\mu_2, \sigma_2^2)$ if $\Delta = 1$

Our task is sometimes to *generate*, but first we have to *estimate* the underlying parameters used in the model. To use the model, we have **5** things to estimate or choose.

$$\Theta = \left(\pi, \mu_1, \sigma_1^2, \mu_2, \sigma_2^2\right)$$

The Gaussian Mixture Model (GMM): 1D Example

The whole model: $X = (1 - \Delta) \cdot X_1 + \Delta \cdot X_2$ We need to estimate:

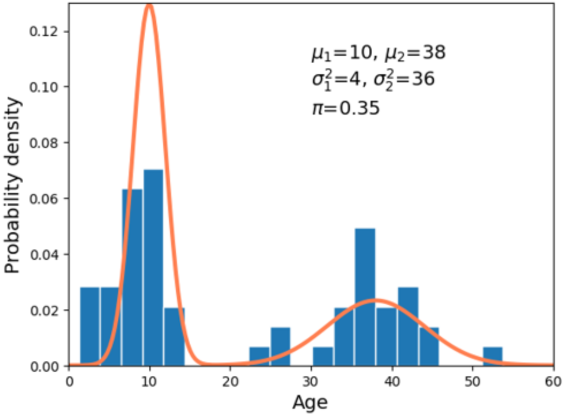
$$\Theta = \left(\pi, \mu_1, \sigma_1^2, \mu_2, \sigma_2^2\right)$$

Assuming we actually *know* all 5 parameters, we can simply write down the full probability density function for our process. If we denote $\phi(x|\mu, \sigma^2)$ as the normal with mean μ and variance σ^2, the model is now

$$f(x|\Theta) = (1 - \pi)\phi(x|\mu_1, \sigma_1^2) + \pi\phi(x|\mu_2, \sigma_2^2)$$

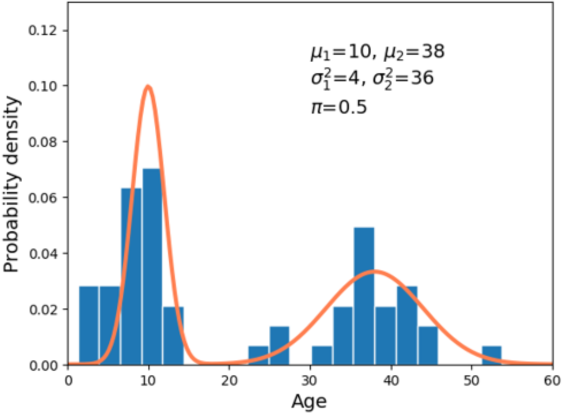
GMM Example: Varying Theta

Here are some pdfs, depending on different choices of the parameter set Θ.



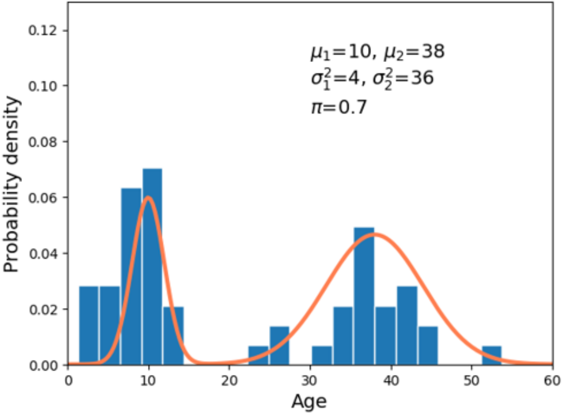
GMM Example: Varying Theta

Here are some pdfs, depending on different choices of the parameter set Θ.



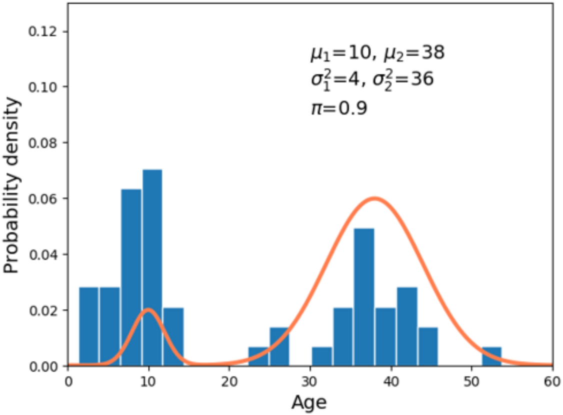
GMM Example: Varying Theta

Here are some pdfs, depending on different choices of the parameter set Θ.



GMM Example: Varying Theta

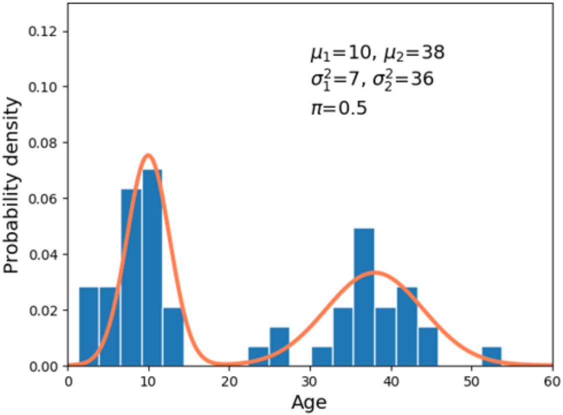
Here are some pdfs, depending on different choices of the parameter set Θ.



This is too many adults: let's go back to $\pi = .5$...

GMM Example: Varying Theta

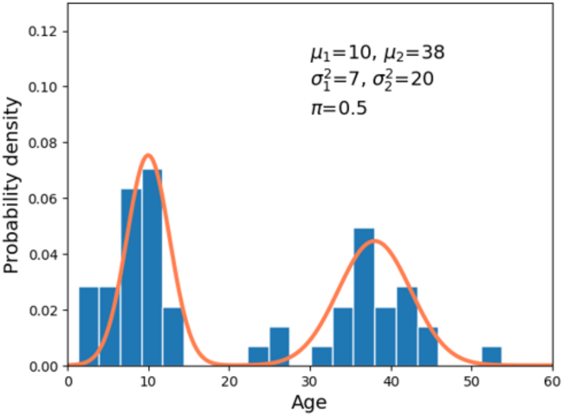
Here are some pdfs, depending on different choices of the parameter set Θ.



Changing σ_1

GMM Example: Varying Theta

Here are some pdfs, depending on different choices of the parameter set Θ.



Changing σ_2

GMM Example: Underview

The whole model: $X = (1 - \Delta) \cdot X_1 + \Delta \cdot X_2$ We need to estimate:

$$\Theta = \left(\pi, \mu_1, \sigma_1^2, \mu_2, \sigma_2^2\right)$$

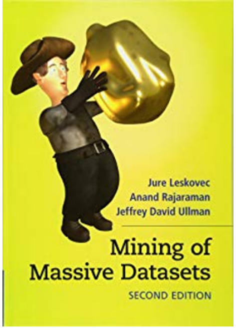
1. It would be **tedious** and hand-wavy to manually try to estimate those parameters, even in one dimension.
2. Note that in 2D, each variance was a 2×2 covariance matrix.

   This problem grows in size *quickly*: 2 Gaussians in 2D is now 9 unknowns (4 per Gaussians plus a mixture probability).

   Next time: some theory on how to estimate the parameters!

Acknowledgments

Some material is adapted/adopted from Mining of Massive Data Sets, by Jure Leskovec, Anand Rajaraman, Jeff Ullman (Stanford University) `http://www.mmds.org`



Special thanks to Tony Wong for sharing his original adaptation and adoption of slide material.