ColumbiaX: Machine Learning
Lecture 16

Prof. John Paisley

Department of Electrical Engineering
& Data Science Institute

Columbia University

focus on this lecture

SOFT CLUSTERING VS HARD CLUSTERING MODELS

which we saw with k-means 2 lectures ago.

Review: K-means clustering algorithm

Given: Data $x_1, \ldots, x_n$, where $x \in \mathbb{R}^d$

Goal: Minimize $\mathcal{L} = \sum_{i=1}^{n} \sum_{k=1}^{K} \mathbb{1}\{c_i = k\} \|x_i - \mu_k\|^2$. * over c_i, μ_k

- Iterate until values no longer changing

assign i^{th} data point to closest cluster

1. Update $\boldsymbol{c}$: For each i, set $c_i = \arg\min_k \|x_i - \mu_k\|^2$
2. Update $\boldsymbol{\mu}$: For each k, set $\mu_k = \left(\sum_i x_i \mathbb{1}\{c_i = k\}\right) / \left(\sum_i \mathbb{1}\{c_i = k\}\right)$

Then when we allocate data points to nearest clusters, we update cluster centroids by simply averaging.

K-means is an example of a *hard clustering* algorithm because it assigns each observation to only one cluster. (after it converges and outputs the final clustering.) And we don't have any uncertainty. **

In other words, $c_i = k$ for some $k \in \{1, \ldots, K\}$. There is no accounting for the "boundary cases" by hedging on the corresponding c_i.

*objective function:

an objective that looks like this where we sum over each data point.
The squared distance to the cluster centroid that it was assigned to
as encoded by the parameters CI.
So XI is assigned to one of capital K clusters and
CI indexes what cluster that is.
So we sum up the total number of square distances.
And that's our objective that we're trying to minimize

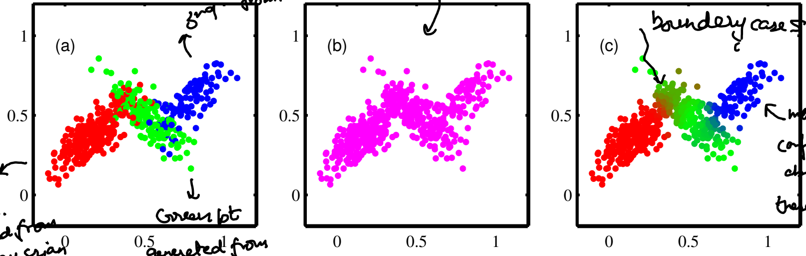
** what change we want to make?

However you could imagine that on the boundary of this clusters for example a data point that is could go either way.
We might want want encode our uncertainty about the clustering of that data point with some sort of a weight.
Whereas the points that are right in the middle of the cluster,
we would want to be more confident about

This leads to concept of a soft clustering algorithm.

SOFT CLUSTERING MODELS

A *soft clustering* algorithm breaks the data across clusters intelligently.



(left) True cluster assignments of data from three Gaussians.

(middle) The data as we see it.

(right) A soft-clustering of the data accounting for borderline cases.

Weighted K-means (soft clustering example)

(considering this in the content of k-means leads to weighted k-means. Modification turns K-means from hard to soft clustering algorithm)

Weighted K-means clustering algorithm

Given: Data $x_1, \ldots, x_n$, where $x \in \mathbb{R}^d$

Goal: Minimize $\mathcal{L} = \sum_{i=1}^{n} \sum_{k=1}^{K} \phi_i(k) \frac{\|x_i - \mu_k\|^2}{\beta}$ over ϕ_i and μ_k

$- \sum_i H(\phi_i)$ But now divide by β.

Conditions: $\phi_i(k) > 0$ and $\sum_{k=1}^{K} \phi_i(k) = 1$. Set parameter $\beta > 0$.

, $H(\phi_i)$ = entropy

- Iterate the following

1. Update ϕ: For each i, update the word allocation weights

$$\phi_i(k) = \frac{\exp\{-\frac{1}{\beta}\|x_i - \mu_k\|^2\}}{\sum_j \exp\{-\frac{1}{\beta}\|x_i - \mu_j\|^2\}}, \quad \text{for } k = 1, \ldots, K$$

if x_i is very far away from μ_k, this value will be small.

2. Update μ: For each k, update μ_k with the *weighted* average

$$\mu_k = \frac{\sum_i x_i \phi_i(k)}{\sum_i \phi_i(k)}$$

each data point weight by the prob. of coming from that cluster.

② Instead of multiplying it by an indicator that says x_i was assigned to cluster k, we multiply it by a weight $\phi_i(k)$

↳ datapoint we are looking at.

→ cluster we are considering.

These ϕs now can be viewed as probability weights. So each value of $\phi_i(k) \geq 0$. And sum of them is equal to 1. (over k.)

→ And so in this case, we've breaking each data point up into different clusters according to how probable we think that cluster is in generating the data point we've looking at.

** For a particular iteration, we look through each of the data points and instead of assigning them to each cluster we assign responsibilities across the clusters according to weight ϕ.

And so in this case, we're breaking each data point up into different clusters according to how probable we think that cluster is in generating the data point we're looking at.

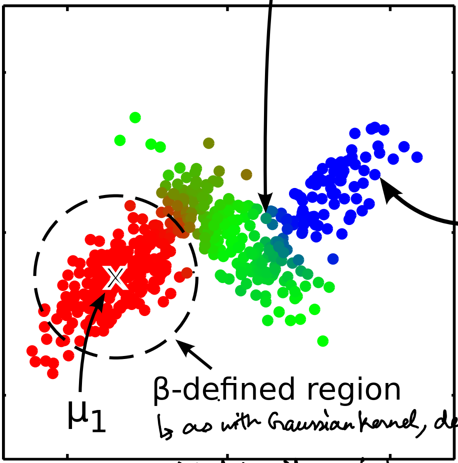
So this is how we take proximity to the cluster centroids into account. To break the responsibility for that given data point across the clusters according to how close that data point is to each cluster.

Difference:

So you can imagine that if these phi's are pure indicators meaning the vector of all zeroes except for a one in the dimension indicating what cluster xi came from.
That would correspond to a hard clustering algorithm and we would simply be summingup all the points in the kth cluster and dividing by the total number of points. Now we are softening that by taking fractions of points and splitting them across clusters

ϕ_i = 0.75 on green cluster & 0.25 blue cluster

The border points are split across clusters to account for the fact that we're less certain which cluster they come from.



ϕ_i = 1 on blue cluster

When ϕ_i is binary, we get back the hard clustering model

↳ as with Gaussian kernel, defines what approximity means. What does it mean to be close and what does it mean to be far away? A parameter we have to set.

Difference k-means and mixture models:

Kmeans:

With soft k-means we took each data point and we, instead of assigning it to one and only one cluster, we broke that data point across clusters using a vector of weights.
And we loosely use the term probability vector,even though there were no probabilistic assumptions.

MIXTURE MODELS

Mixture Models:

So now we’re gonna look at a modeling framework where we do make these sorts of probabilistic assumptions, by defining probability Distributions.

Probabilistic vs non-probabilistic soft clustering

The weight vector ϕ_i is *like* a probability of x_i being assigned to each cluster.

A **mixture model** is a probabilistic model where ϕ_i actually *is* a probability distribution according to the model.

So, c_i is going to come back into our models, and we're going to define a probability distribution.

Mixture models work by defining:
(We explicitly define a probability distribution on it.)

- A prior distribution on the cluster assignment indicator c_i
- A likelihood distribution on observation x_i given the assignment c_i

So, we're now using probabilistic definition to put these constraints on our parameters.

Intuitively we can connect a mixture model to the Bayes classifier:

- Class prior → cluster prior. * This time, we *don't* know the "label" **
- Class-conditional likelihood → cluster-conditional likelihood **

* It's the prior on a particular data point coming from a particular cluster, except this time now, we don't know the label for each observation.

**

Previously the cluster, each cluster was like a class, and we knew the label for each observation, and therefore we knew which cluster it came from.
Now we don't know that information

*** Conditional distribution similarility:

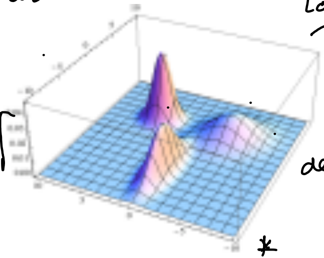
And similarly with the bayes classifier we had the class conditional distributions,so given what class it comes from we have a specific class distribution on the data. Now we have cluster conditional distributions, sp condition on a data point coming from a particular cluster we have.
A specific distribution on that data point for that, coming from that cluster.

MIXTURE MODELS

- generative model, we're going to define a probability distribution on the data and try to learn parameters of that distribution.

weighted combination of simpler distributions.

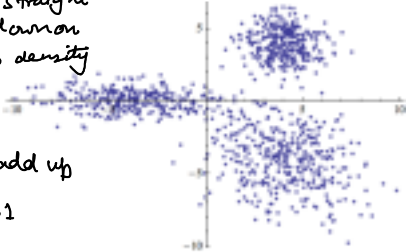
density in $\mathbb{R}^2$



Looking straight down on this density

density add up to 1

*



(a) A probability distribution on $\mathbb{R}^2$. (b) Data sampled from this distribution.

And now we want to propose a Gaussian mixture model. And then learn the resulting means, covariances and mixing weights on those 3 different Gaussians.

Before introducing math, some key features of a mixture model are:

1. It is a generative model (defines a probability distribution on the data)
2. It is a weighted combination of simpler distributions.
   - Each simple distribution is in the same distribution family (i.e., a Gaussian).
   - The "weighting" is defined by a discrete probability distribution.

*These three distributions are actually 3 different Gaussian distributions with different means and covariances. We weight each of these and put them together according to the probability of particular Gaussian to get our final density.

to define ?

define a generative process. (Same with all bayesian models. If we want to define what's it doing we have hypothesise how data is generated from that model.)

Generating data from a mixture model

Data: $x_1, \ldots, x_n$, where each $x_i \in \mathcal{X}$ (can be complicated, but think $\mathcal{X} = \mathbb{R}^d$)

Model parameters: A K-dim distribution π * and parameters $\theta_1, \ldots, \theta_K$.

on each of those clusters. 1st cluster parameters

Generative process: For observation number $i = 1, \ldots, n$,

1. Generate cluster assignment: $c_i \overset{iid}{\sim} \text{Discrete}(\pi) \Rightarrow \text{Prob}(c_i = k|\pi) = \pi_k$.
2. Generate observation: $x_i \sim p(x|\theta_{c_i})$.

class of probability distributions.

Some observations about this procedure:

- First, each x_i is randomly assigned to a cluster using distribution π.
- c_i indexes the cluster assignment for x_i
  - This picks out the index of the parameter θ used to generate x_i.
  - If two x's share a parameter, they are clustered together. (Generated from the same probability distribution.)

* if we assume there are k different clusters, we have k dimensional probability distribution on each of those clusters, and also the parameters for the clusters.

** Every single data point that we are going to assume is generated independently with same discrete π distribution.

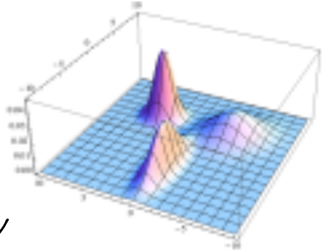
*** Meaning that the probability that we assign any given data point to the k^{th} cluster given π is equal to π_K.

- Then, when we assign it to the cluster, we pick the parameter to generate the data point according to the indexing of the cluster. And so we use the same exact probability distribution.

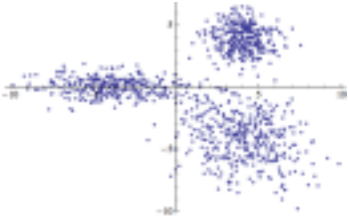
Distribution of p:

So we're going to assume that P is the same for all clusters. So it's, for example, a multivariate Gaussian or something more complicated than that. The only thing that's differing across clusters is what parameter we use for that cluster. ($\theta - \mu, \Sigma$ for each)

MIXTURE MODELS (Gaussian setting to see the impact of weights only.)



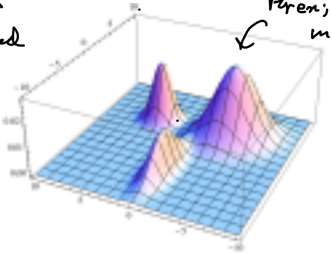
(a) Uniform mixing weights



(b) Data sampled from this distribution.

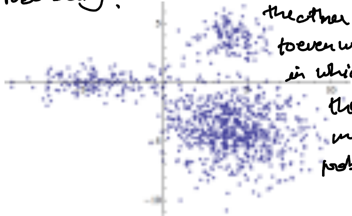
same gaussian but we have changed weights

Here; this cluster gets more probability.



(c) Uneven mixing weights

where we have more data from one cluster than the other as opposed to even weights, in which case they're more probable.



(d) Data sampled from this distribution.

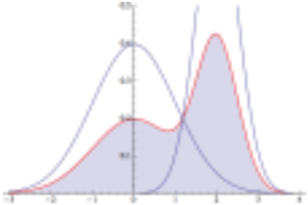
→ Gaussians define the region & the weights define how dense the data is in each region.

Gaussian mixture models

ILLUSTRATION

Gaussian mixture models are mixture models where $p(x|\theta)$ is Gaussian.

Mixture of two Gaussians

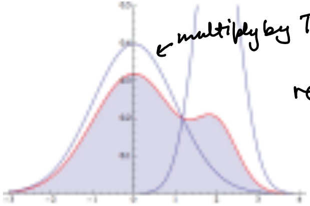


The red line is the density function.

$$\pi = [0.5,\ 0.5]$$

$$\begin{aligned}(\mu_1, \sigma_1^2) &= (0, 1)\\ (\mu_2, \sigma_2^2) &= (2, 0.5)\end{aligned}$$

Influence of mixing weights



← multiply by π

red line is the final.

The red line is the density function.

$$\pi = [0.8,\ 0.2]$$

$$\begin{aligned}(\mu_1, \sigma_1^2) &= (0, 1)\\ (\mu_2, \sigma_2^2) &= (2, 0.5)\end{aligned}$$

GAUSSIAN MIXTURE MODELS (GMM)

The model

Ok.

Parameters: Let π be a K-dimensional probability distribution and (μ_k, Σ_k) be the mean and covariance of the kth Gaussian in $\mathbb{R}^d$.

Generate data: For the ith observation,

1. Assign the ith observation to a cluster, $c_i \sim \text{Discrete}(\pi)$
2. Generate the value of the observation, $x_i \sim N(\mu_{c_i}, \Sigma_{c_i})$

We've defined the hypothesis of how the data got to us by defining the generative model. Now, that defines a joint likelihood on the data given some parameters.

Definitions: $\boldsymbol{\mu} = \{\mu_1, \ldots, \mu_K\}$ and $\boldsymbol{\Sigma} = \{\Sigma_1, \ldots, \Sigma_k\}$.

Goal: We want to learn π, $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$.

Once we have the data, our goal is go back and infer what are the parameters that could have generated that data. (Inference problem)

Maximum likelihood

Objective: Maximize the likelihood over model parameters π, $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ by treating the c_i as auxiliary data using the EM algorithm.

↳ additional hidden data that we introduce. So we want to integrate out c_i.

$$p(x_1, \ldots, x_n|\pi, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \prod_{i=1}^{n} p(x_i|\pi, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \prod_{i=1}^{n} \sum_{k=1}^{K} p(x_i, c_i = k|\pi, \boldsymbol{\mu}, \boldsymbol{\Sigma})$$

i.i.d assumption simplification. no c_i here (we integrated it out.)

The summation over values of each c_i "integrates out" this variable.

why?

c_i is Θ_2 (last lecture)

We can't simply take derivatives with respect to π, μ_k and Σ_k and set to zero to maximize this because there's no closed form solution. (can't solve it exactly).

We could use gradient methods, but EM is cleaner.

(because we can get closed form updates.)

Goal to maximize the joint likelihood of data given only π, μ & Σ, where we have summed the parameter c_i out of the model.

* So CI, the cluster assignment now is going to be
the additional hidden data that we introduced, and then do.
So we want to integrate out CI.
Remember, from the Algorithm that we discussed last time.
What that means is that we wanted to maximize the likelihood
of all of the data, x1 through xn, given pi, mu, and sigma.

* *

But now we integrate or in this case,
because CI can only take a discreet and finite number of values.
We sum over all those possible values to integrate them out.

- Joint likelihood of the i^{th} observation and cluster assignment which cluster the i^{th} observation came from.

Q: Why not instead just include each c_i and maximize $\prod_{i=1}^{n} p(x_i, c_i|\pi, \boldsymbol{\mu}, \boldsymbol{\Sigma})$ since (we can show) this is easy to do using coordinate ascent?

*

A: We would end up with a hard-clustering model where $c_i \in \{1, \ldots, K\}$. Our goal here is to have soft clustering, which the sum effectively does.

EM and the GMM

We will not derive everything from scratch. However, we can treat $c_1, \ldots, c_n$ as the auxiliary data that we integrate out. (latent data, missing data in our model, that we want to integrate over.)

Therefore, we use EM to

marginal likelihood where c_i is gone

log of this augmented / extended joint likelihood.

$$\text{maximize} \quad \sum_{i=1}^{n} \ln p(x_i|\pi, \boldsymbol{\mu}, \boldsymbol{\Sigma}) \quad \text{by using} \quad \sum_{i=1}^{n} \ln p(x_i, c_i|\pi, \boldsymbol{\mu}, \boldsymbol{\Sigma}).$$

Let's look at the outlines of how to derive this.

*

Because when we do that, when we write out the augmented joint likelihood,
where we include CI as a parameter we want to do a point estimate over,
we take the log of this, and we take derivatives and solve.
What we would find is that we can optimize that very easily,
we can get closed form updates for every parameter in the model including CI.
So that is something that can be done

But,

what we would find by doing that Is we would end up with a hard clustering model.
Meaning that, when we wanted to update a particular cluster assignment,
CI, the maximizing that
CI would lead to assigning the data point to the most probably cluster.
So the maximum likelihood solution, or the map solution,
actually, for CI, is to assign the ith data point to the most probable cluster.
And so we would have a hard clustering algorithm.

Goal:

Our goal here is to integrate out the effect of the cluster assignment
Which we're going to see when we do that in the Framework.
It's going to lead to a soft clustering algorithm that
we anticipate should be more accurate and should be better than the hard clustering algorithm because we've integrated out one of the unknowns in our model.
So we have fewer potential local optimal solutions that we could converge to.

THE EM ALGORITHM AND THE GMM (How the equation is modified for this model structure?)

From the last lecture, the generic EM objective is

We introduced the additional variable to our model θ_2 and integrate it out. *

And we do a 2-step procedure:

$$\ln p(x|\theta_1) = \underbrace{\int q(\theta_2) \ln \frac{p(x, \theta_2|\theta_1)}{q(\theta_2)}\, d\theta_2}_{\mathcal{L}} + \underbrace{\int q(\theta_2) \ln \frac{q(\theta_2)}{p(\theta_2|x, \theta_1)}\, d\theta_2}_{KL(q\,\|\,p)}$$

maximize the log of probability of the x given parameter θ_1

Conditional posterior distribution.

The EM objective for the Gaussian mixture model is

Take this equality, and we just write it using the Gaussian mixture model parameters:

$$\sum_{i=1}^{n} \ln p(x_i|\pi, \boldsymbol{\mu}, \Sigma) \;=\; \sum_{i=1}^{n} \sum_{k=1}^{K} q(c_i = k) \ln \frac{p(x_i, c_i = k|\pi, \boldsymbol{\mu}, \Sigma)}{\cancel{q(c_i = k)}} \; +$$

treat as this

**

$$\underbrace{\sum_{i=1}^{n} \sum_{k=1}^{K} q(c_i = k) \ln \frac{q(c_i = k)}{p(c_i|x_i, \pi, \boldsymbol{\mu}, \Sigma)}}_{i\text{th KL divergence}}$$

In next slide, we are going to ignore it, not involve the parameters we're interested in.

θ_1 corresponds to

θ_2 corresponds to c_i

Because c_i is discrete, the integral becomes a sum.

remember the procedures were to set Q equal to P.
Given that updated Q distribution,
calculate this expectation, and then maximize this expectation over theta1.
And theta 2 is gone because it was integrated out.

So, the log of the likelihood of all of the data, given pi mu and sigma
because the data are assumed to be IID, that turns into the log of the product,
which turns into the sum of the log of each individual likelihood.
So we have a sum over all of the data of the log of the probability of each point
given our Gaussian mixture model parameters.

- And because ci is a discrete variable the integral is going to turn in to a sum.
  So when you have continuous
  variables that you integrate when you have discrete ones you sum.

so this integral, overall, the values that theta_2 can take turns into a sum
overall the values that CI can take.

EM Setup (one iteration)

you do this for every k

(Set KL divergence = 0)

First: Set $q(c_i = k) \;\Leftarrow\; p(c_i = k|x_i, \pi, \boldsymbol{\mu}, \boldsymbol{\Sigma})$ using Bayes rule:

likelihood of the i^{th} observation given that it was assigned to cluster.

$$p(c_i = k|x_i, \pi, \boldsymbol{\mu}, \boldsymbol{\Sigma}) \propto p(c_i = k|\pi)p(x_i|c_i = k, \boldsymbol{\mu}, \boldsymbol{\Sigma})$$

conditional posterior

assigning i^{th} datapoint to k^{th} cluster

We can solve the posterior of c_i given π, $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$:

$$q(c_i = k) = \frac{\overset{*}{\pi_k} \overset{**}{N}(x_i|\mu_k, \Sigma_k)}{\sum_j \pi_j N(x_i|\mu_j, \Sigma_j)} \implies \phi_i(k)$$

serving same purpose as weighted k-means algo. Now we have a firm probabilistic interpretation of what we're doing.

II^{nd} part after we update each of these key distributions go to

E-step: Take the expectation using the updated q's

calculate the expectation (L):

sum over each observation,

$$\underset{(\alpha)}{Q} = \sum_{i=1}^{n} \sum_{k=1}^{K} \phi_i(k) \ln \overset{***}{p}(x_i, c_i = k|\pi, \mu_k, \Sigma_k) \;+\; \text{constant w.r.t. } \pi, \boldsymbol{\mu}, \boldsymbol{\Sigma}$$

M-step: Maximize Q (α) with respect to π and each μ_k, Σ_k.

weight we assign to k^{th} cluster.

we can expand this, write out the Gaussian form of it. And then get this $f^{(n)}$ that we want to maximise.

what does this mean?

* The prior probability of assigning a data point to cluster k is just equal to π_k.

** The likelihood of i^{th} observation coming from cluster k is equal to Gaussian
→ where we evaluate the likelihood of x_i given the μ & Σ of k^{th} cluster.

→ Normalization

And then to turn that in to a probability distribution we simply
normalize the right-hand side.
So this is a proportionality, to turn this into an equality we normalize this thing
by dividing over the sum of all possible values that CI can take,
meaning we sum this numerator over all values k.
So we take this divided by the sum over all possible values for
the cluster assignment.

*** Where we evaluate the log of the likelihood of
the data coming from each cluster.
So, that's this term the log and
the likelihood of the data point coming from kth cluster
plus the log of the prior probability of it coming from the kth cluster.
So that's equivalent to the log of the joint likelihood
of the observation coming from the kth cluster.
And it coming from, any point coming from the kth cluster.

?

joint?

M-STEP CLOSE UP

*the log of likelihood of data given π, μ and σ where we integrate out c.

Aside: How has EM made this easier?

$\sum_{i=1}^{n} \ln p(x_i | \pi, \mu, \Sigma)$

Original objective function:

how?

$$\mathcal{L} = \sum_{i=1}^{n} \ln \underbrace{\sum_{k=1}^{K} p(x_i, c_i = k | \pi, \mu_k, \Sigma_k)}_{\text{integrate out } C \ \text{I}^{*}} = \sum_{i=1}^{n} \ln \sum_{k=1}^{K} \pi_k N(x_i | \mu_k, \Sigma_k).$$

The log-sum form makes optimizing π, and each μ_k and Σ_k difficult.

Using EM here, we have the M-Step:

Before we had a log sum, we turned it into a sum of log. * **

$$Q = \sum_{i=1}^{n} \sum_{k=1}^{K} \phi_i(k) \underbrace{\{\ln \pi_k + \ln N(x_i | \mu_k, \Sigma_k)\}}_{\ln p(x_i, c_i = k | \pi, \mu_k, \Sigma_k)} + \text{constant w.r.t. } \pi, \boldsymbol{\mu}, \boldsymbol{\Sigma}$$

The sum-log form is easier to optimize. We can take derivatives and solve.

** And this log sum,
this sum appearing inside of the log is what makes it difficult.
If you took the derivative of this with respect to mu or
sigma or pi you would see that this sum inside is
what's the thing that's making everything complicated for us.

why?

*** If we now take the derivative of this and try to maximize it with respect to pi,
mu and sigma, we'll find that it's very easy to do that.
And so that's what we do.
We actually take the derivative of this thing with respect to mu, the means,
the covariances, and also pi,
conditioned with the constraint that pi is a probability distribution.
And we maximize this function with respect to those three unknown parameters.
And we'll find that we get a closed form solution.

Algorithm: Maximum likelihood EM for the GMM

Given: $x_1, \ldots, x_n$ where $x \in \mathbb{R}^d$

Goal: Maximize $\mathcal{L} = \sum_{i=1}^{n} \ln p(x_i|\pi, \boldsymbol{\mu}, \boldsymbol{\Sigma})$.

Log of marginal likelihood of each data point over π, μ, Σ, where there is no cluster assignment at all to be found because it is integrated it out.

- Iterate until incremental improvement to $\mathcal{L}$ is "small"

1. **E-step**: For $i = 1, \ldots, n$, set *

(Prob. of i^{th} data pt. coming from k^{th} cluster.)

$$\phi_i(k) = \frac{\pi_k N(x_i|\mu_k, \Sigma_k)}{\sum_j \pi_j N(x_i|\mu_j, \Sigma_j)}^{**}, \quad \text{for } k = 1, \ldots, K$$

how did you get these expressions?

2. **M-step**: For $k = 1, \ldots, K$ (each cluster), define $n_k = \sum_{i=1}^{n} \phi_i(k)$ and update the values

Then we update 3 parameters of the model:

(prior prob. of the k^{th} cluster)

$$\pi_k = \frac{n_k}{n}, \quad \mu_k = \frac{1}{n_k}\sum_{i=1}^{n} \phi_i(k) x_i \quad \Sigma_k = \frac{1}{n_k}\sum_{i=1}^{n} \phi_i(k)(x_i - \mu_k)(x_i - \mu_k)^T$$

soft-covariance matrix

Comment: The updated value for μ_k is used when updating Σ_k.

empirical distribution on the clusters acc. to our assignments (our soft assignments)

1. We perform soft-clustering of the data by taking each data point and calculate conditional posterior probability of that data point coming from each of the k different clusters.

And we can do that this way **, where calculate the prior probability of the k^{th} cluster times the likelihood of the observation given it comes from k^{th} cluster. And then divide by the sum of that over each cluster.

o sum of the expected no. of points that we're going to see coming from the k^{th} cluster acc. to the current iteration

oo

For the mean, we then take each data point.
We multiply it by the probability of that data point coming from the Kth cluster.
We sum it up and
divide by the expected total number points coming from that cluster.

Mean calculation same/difference from k means.

So notice that this actually is identical to weighted k means.
This is exactly the same as weighted k means.
The only difference is the way that we calculated this phi.
So this phi, the way we calculated phi with the Gaussian mixture model,
used this equation.
If you look at the way we calculated phi for weighted k means,
it used a slightly different equation.
But then once we have the soft clustering of the ith observation
of each observation, we can calculate the mean exactly in the same way.

[what is difference in phi)

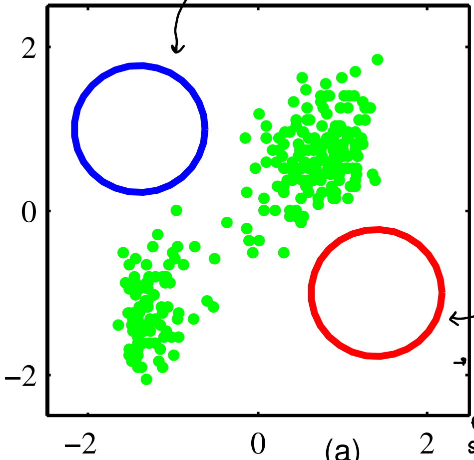
Covariance matrix

But then in addition to that with the Gaussian mixture model,
we get a covariance matrix for
the Kth cluster, which is also like the weighted
empirical covariance of the data coming from the Kth cluster.
So from mu k here, this is the mean of the Kth cluster.
We use the updated mu here.
We calculate this outer product.
We multiply it by the probability that the ith point came from the Kth cluster
to begin with.
Sum that up over each data point.
And then divide by the expected total number of points.
So this is a soft covariance matrix.

GAUSSIAN MIXTURE MODEL: EXAMPLE RUN

In this case, we define 2 Gaussians.

IInd Gaussian



(a)

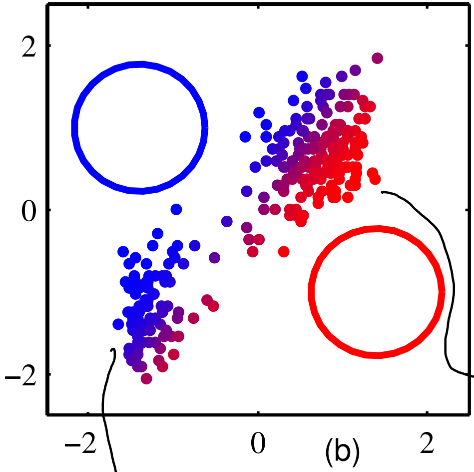
A random initialization

Ist Gaussian

→ Circle indicates spherical covariance for each of these Gaussians.

Also, they are equally probable, the wt. is not being shown on the prob. of each of these clusters.

Gaussian mixture model: Example run



Iteration 1 (E-step)

Assign data to clusters

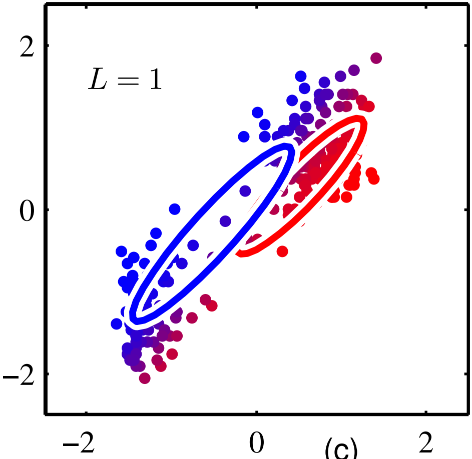
We take data point and calculate the posterior prob. of that point coming from each of these clusters.

color code based on the weight of prob. so this point is entirely red to indicate its almost entirely coming from this cluster.

in the middle, its 50/50 on either of these 2 clusters.

Diff. from k means. Everything on 1 side, entirely red, and everything on other side entirely blue.

Gaussian mixture model: Example run



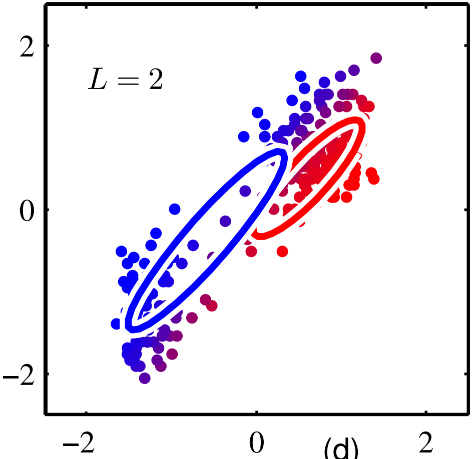
(c)

Iteration 1 (M-step)

Update the Gaussians

We calculate the empirical weighted mean and weighted covariance.

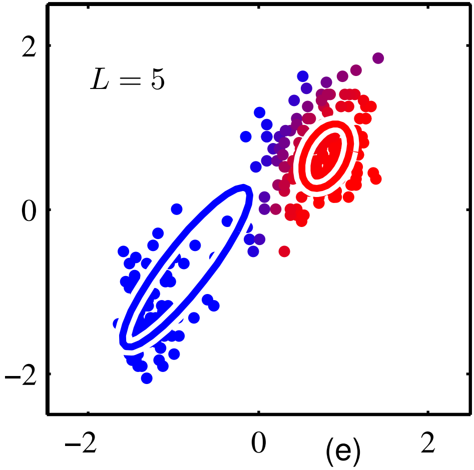
GAUSSIAN MIXTURE MODEL: EXAMPLE RUN



(d)

Iteration 2

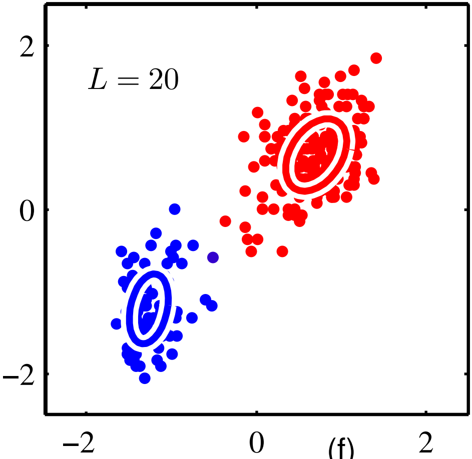
Assign data to clusters and update the Gaussians



Iteration 5 (skipping ahead)

Assign data to clusters
and update the Gaussians

Gaussian mixture model: Example run



(f)

Iteration 20 (convergence)

Assign data to clusters
and update the Gaussians

[Turns out for this particular dataset, we're very confident in our clustering.]

And we also get a probability of each of the Gaussians. So, the Gaussian has a certain prob. based on the empirical proportion of data coming from that cluster.

GMM AND THE BAYES CLASSIFIER [Intuitively they feel similar, but there are significant differences.]

with k clusters — where we had a class-specific multi-variate Gaussian density

The GMM feels a lot like a K-class Bayes classifier, where the label of x_i is

class prior — probability of data point coming from class k times the likelihood of that particular data point.

$$\text{label}(x_i) = \arg\max_k \; \pi_k N(x_i|\mu_k, \Sigma_k).$$

- π_k = class prior, and $N(\mu_k, \Sigma_k)$ = class-conditional density function.
- Also, We learned π, $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ using maximum likelihood here too. (And had similar looking updates for these parameters, except we didn't have an algorithm in that case. We just simply gave an equation for MLE for π, μ, Σ.)

For the Bayes classifier, we could find π, $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ with a single equation because the class label was *known*. Compare with the GMM update: (where we have an algorithm.)

$$\pi_k = \frac{n_k}{n}, \quad \mu_k = \frac{1}{n_k}\sum_{i=1}^{n}\phi_i(k)x_i \quad \Sigma_k = \frac{1}{n_k}\sum_{i=1}^{n}\phi_i(k)(x_i-\mu_k)(x_i-\mu_k)^T$$

They're almost identical. But* since $\phi_i(k)$** is changing we have to update these values. With the Bayes classifier, "ϕ_i" encodes the label, so it was known.

* But what's changed:

The key thing that's changed is that with bayes classifier we already had the cluster assignments. So the clusters were like the classes and we had already assigned each of the data points to their specific classes. Because we had labels for them in our dataset. So we didn't need to ask which cluster/class any data point came from. We simply assigned it to its specific class.

GMM:

Now, with clustering, with this GMM which has a very similar form. We suddenly don't know for any datapoint which cluster it came from.

What that tells you:

↳ Conditioned on the cluster assignment we can do MLE exactly. But when we don't have the cluster assignment, we have to iterate between updating the cluster assignment and then updating the model parameters.

We have to run an iterative algorithm because we can't optimize both simultaneously.

** $\phi_i(k)$ → is like encoding a probability of what label the i^{th} data point has, if you were to view a cluster like a label. For that data point we're infering both the labels and class-specific densities simultaneously.

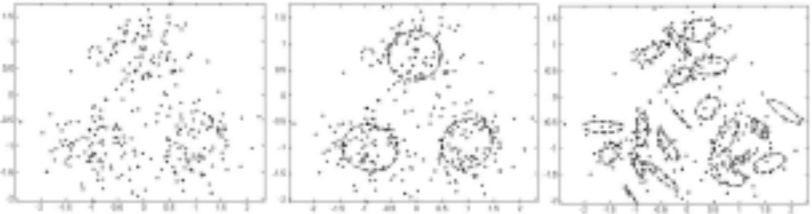
?

Although, it's not typical to think of this sort of unsupervised mixture modelling as a classification problem. We don't ever usually discuss the clusters as being classes but intuitively the math is very similar.

CHOOSING THE NUMBER OF CLUSTERS

no. of clusters you give it is the no. of clusters it learns (similarity with kmeans)

overfitting with EM.



Maximum likelihood for the Gaussian mixture model can overfit the data. It will learn as many Gaussians as it's given.

There are a set of techniques for this based on the Dirichlet distribution.

A Dirichlet prior is used on π which encourages many Gaussians to disappear (i.e., not have any data assigned to them). ?

EM FOR A GENERIC MIXTURE MODEL

[Generalized to mixture models using different distributions besides multi-variate Gaussians.]

Algorithm: Maximum likelihood EM for mixture models

Given: Data $x_1, \ldots, x_n$ where $x \in \mathcal{X}$

some parameter set

cluster specific likelihood. *

iid assumption

Goal: Maximize $\mathcal{L} = \sum_{i=1}^{n} \ln p(x_i|\pi, \boldsymbol{\theta})$, where $p(x|\theta_k)$ is problem-specific.

mixing weights

- Iterate until incremental improvement to $\mathcal{L}$ is "small"

1. **E-step**: For $i = 1, \ldots, n$, set

prior probability of a point coming from k^{th} cluster

likelihood of i^{th} data point coming from the k^{th} cluster **

conditional posterior probability of i^{th} observation coming from the k^{th} cluster.

$$\phi_i(k) = \frac{\pi_k \, p(x_i|\theta_k)}{\sum_j \pi_j \, p(x_i|\theta_j)}, \quad \text{for } k = 1, \ldots, K$$

similar from before: [sum the probabilities of all of the data coming from that cluster. Get the expected no. of points we're gonna see from that cluster.]

2. **M-step**: For $k = 1, \ldots, K$, define $n_k = \sum_{i=1}^{n} \phi_i(k)$ and set

prior prob. of any pt coming from k^{th} cluster.

parameter to the distribution for k^{th} cluster.

likelihood of i^{th} pt given θ. ***

$$\pi_k = \frac{n_k}{n}, \qquad \theta_k = \arg\max_{\theta} \sum_{i=1}^{n} \phi_i(k) \ln p(x_i|\theta)$$

[same exact thing]

Comment: Similar to generalization of the Bayes classifier for any $p(x|\theta_k)$.

* of x given the k^{th} cluster is going to be generic. With the Gaussian mixture model, this was a multivariate Gaussian. And θ_k was k^{th} mean and k^{th} covariance for that Gaussian.
Now this is any prob. distribution and θ_k is the set of parameters for that distribution.

** This is now different from before. Before this was multivariate Gaussian, now it's whatever distribution we've defined on x_i according to the problem we're looking at.

*** So for example, before this was a Gaussian, where we wanted to maximize this thing over the mean and the covariance.
And we took the derivative and we found that we could do that in closed form.

Now, depending on what this distribution is, we might need to run an algorithm to update to maximize this, or maybe we can take the derivatives and set to zero and solve.
But we'll have a different equation given the model that we're considering.
Given the densities that the likelihoods that we're using for the model we're considering.