Mixture distributions

CPSC 440/550: Advanced Machine Learning

cs.ubc.ca/~dsuth/440/23w2

University of British Columbia, on unceded Musqueam land

2023-24 Winter Term 2 (Jan–Apr 2024)

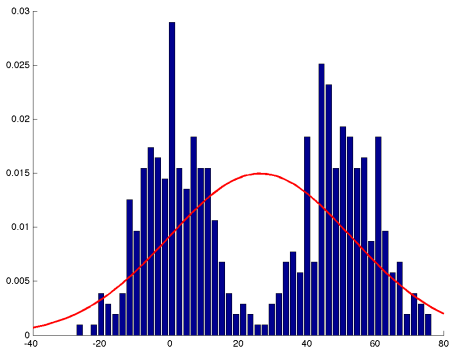
Last time: Exponential families

- Have sufficient statistics and canonical parameters
- Maximimum likelihood becomes moment matching; always have conjugate priors
- Can build discriminative models by using canonical parameter $s(x) = w^{\mathsf{T}} x$
- Many things (but not everything!) are exponential families
  - Today: some things that aren't

Outline

1 Gaussian for Multi-Modal Data

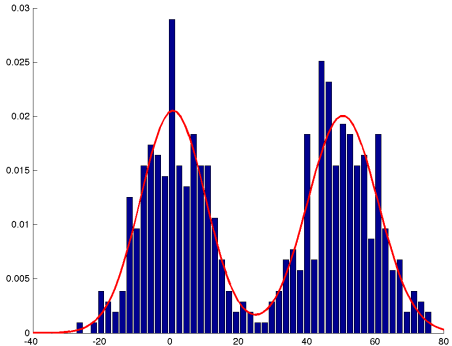
- One major drawback of Gaussian is that it is uni-modal
  - It gives a terrible fit to data like this:



- How can we fit this data?
- Could use an exp. family, but only by harcoding possible mode locations in $s(x)$
- We'll want something more general...

2 Gaussians for Multi-Modal Data

- We can fit this data by using two Gaussians



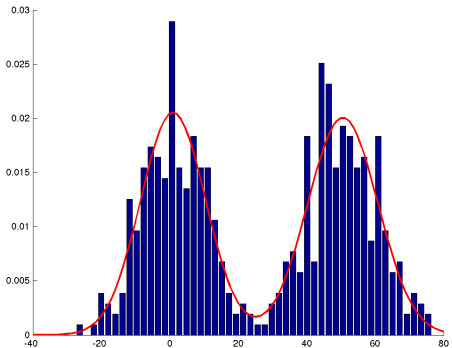
- Half the samples are from Gaussian one, half are from Gaussian two

Mixture of Gaussians

- Our probability density in this example is given by

$$p(x \mid \mu_1, \mu_2, \Sigma_1, \Sigma_2) = \frac{1}{2}\underbrace{\mathcal{N}(x \mid \boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)}_{\text{pdf of Gaussian 1}} + \frac{1}{2}\underbrace{\mathcal{N}(x \mid \boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2)}_{\text{pdf of Gaussian 2}},$$

  - We need the $\frac{1}{2}$s for it to integrate to 1



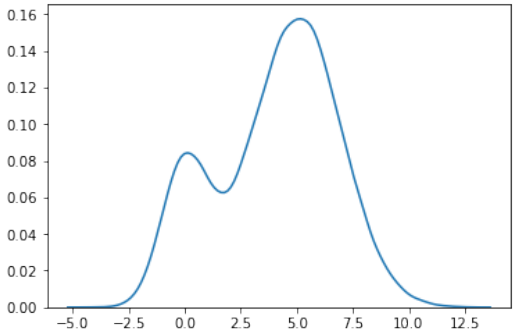
Mixture of Gaussians

- If data comes from one Gaussian more often than the other, we could use

$$p(x \mid \mu_1, \mu_2, \Sigma_1, \Sigma_2, \pi_1, \pi_2) = \pi_1 \underbrace{\mathcal{N}(x \mid \boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)}_{\text{pdf of Gaussian 1}} + \pi_2 \underbrace{\mathcal{N}(x \mid \boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2)}_{\text{pdf of Gaussian 2}},$$

where π_1 and π_2 are non-negative and sum to 1

- π_1 is "probability that we take a sample from Gaussian 1"

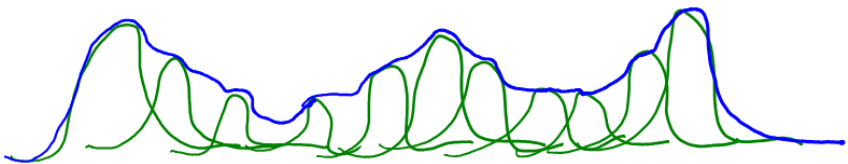


Mixture of Gaussians

- In general we might have a mixture of k Gaussians with different weights

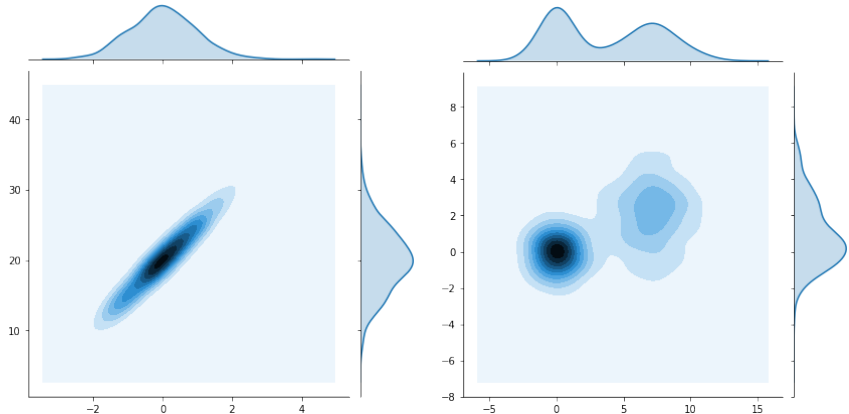
$$p(x \mid \mu, \Sigma, \pi) = \sum_{c=1}^{k} \pi_c \underbrace{\mathcal{N}(x \mid \boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)}_{\text{pdf of Gaussian } c}$$

  - π_c are categorical distribution parameters (non-negative and sum to 1).
  - If k is large, can model complicated densities with Gaussians (like RBFs)
  - "Universal approximator" if $k \to \infty$
    - Can model any continuous density on a compact set



Mixture of Gaussians

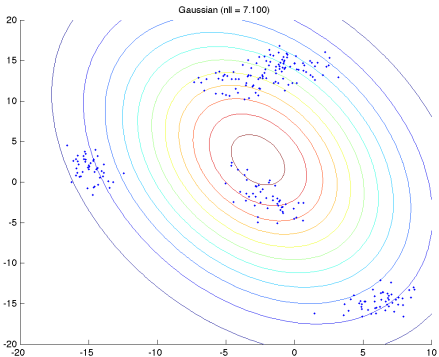
- Gaussian vs. mixture of 2 Gaussian densities in 2D:

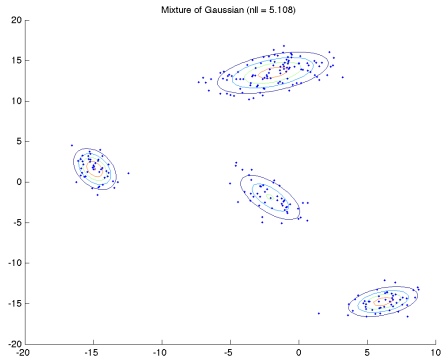


- Marginals will also be mixtures of Gaussians

Mixture of Gaussians

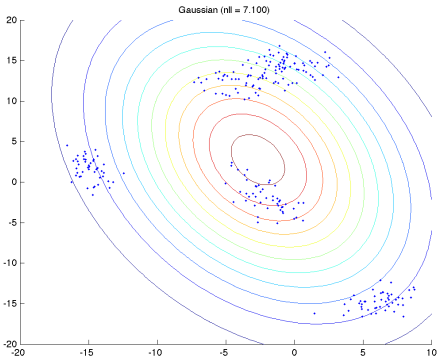
- Gaussian vs. mixture of 4 Gaussians for 2D multi-modal data:

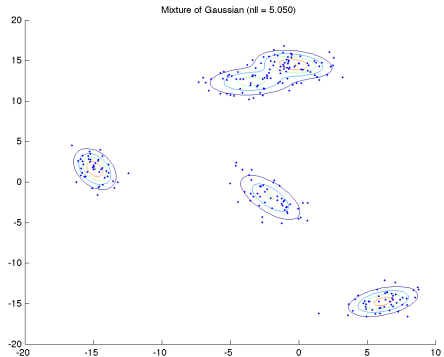




Mixture of Gaussians

- Gaussian vs. mixture of 5 Gaussians for 2D multi-modal data:





Latent-Variable Representation of Mixtures

- For inference/learning in mixture models, we often introduce variables $z^{(i)}$.
  - Each $z^{(i)}$ is a categorical variable in $\{1, 2, \dots, k\}$ when we have k mixtures
  - The value $z^{(i)}$ represents "what mixture this example came from"
  - We do not observe the $z^{(i)}$ values (called latent variables)

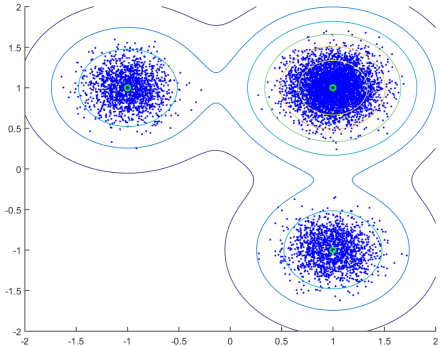
- Why this interpretation of "each x^i comes from one Gaussian"?
  - Consider a model where $p(z = c) = \pi_c$, and $x \mid (z = c) \sim \mathcal{N}(\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)$
  - Now marginalize over the $z^{(i)}$ in this model:

$$p(x \mid \mu, \Sigma, \pi) = \sum_{c=1}^{k} p(x, z = c) = \sum_{c=1}^{k} p(z = c) p(x \mid z = c)$$
$$= \sum_{c=1}^{k} \pi_c \mathcal{N}(x \mid \boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)$$

which is the pdf of the mixture of Gaussians model

Ancestral sampling in mixture of Gaussians

- Generating samples with ancestral sampling in the latent variable representation:
  1. Sample cluster z based on prior probabilities π_c (categorical distribution)
  2. Sample example x based on mean μ_z and covariance Σ_z of Gaussian z



Inference for Gaussian mixtures

- Marginalization and computing conditionals is also easy
- Computing the marginal $p(z \mid x)$, or finding its mode, is easy (next slide)
- Finding the mode for x in Gaussian mixtures is NP-hard

- We usually fit these models with expectation maximization (EM; soon)
  - An optimization method that gives closed-form updates for this model
- Choosing k: domain knowledge, test set likelihood, or marginal likelihood.

Inference Task: Computing Responsibilities

- Consider computing probability that example i came from mixture c
  - We call this the responsibility of mixture c for example i:

$$
\begin{aligned}
r_c^{(i)} &= p(z = c \mid x^{(i)}) \\
&= \frac{p(z = c, x^{(i)})}{p(x^{(i)})} \\
&= \frac{p(z = c, x^{(i)})}{\sum_{c'=1}^{k} p(z' = c, x^{(i)})} \\
&= \frac{p(z = c)\, p(x^{(i)} \mid z = c)}{\sum_{c'=1}^{k} p(z = c')\, p(x^{(i)} \mid z = c')} \\
&= \frac{\pi_c \mathcal{N}(x^{(i)} \mid \boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)}{\sum_{c'=1}^{k} \pi_{c'} \mathcal{N}(x^{(i)} \mid \boldsymbol{\mu}_{c'}, \boldsymbol{\Sigma}_{c'})}
\end{aligned}
$$

(we know all these values)

  - Avoid underflow in computation with log-space: bonus slides
- Thinking of mixture components as clusters, this is probability of being in cluster c

Notation Alert: π vs. z vs. r (MEMORIZE)

- In mixture models, many people confuse the quantities π, z, and r
  - Vector π has k elements in $[0, 1]$ and summing up to 1
    - Number π_c is the "prior" probability that an example is in cluster c
    - This is a parameter (we learn it from data)
  - Matrix $\mathbf{R}$ is an $n \times k$ matrix, summing to 1 across rows
    - Number $r_c^{(i)}$ is the "posterior" probability that example i is in cluster c
    - Computing these values is an inference task (assumes known parameters)
  - Vector $\mathbf{z}$ has n elements in $\{1, 2, \ldots, k\}$
    - Category $z^{(i)}$ is the actual mixture/cluster that generated example i
    - This is a nuisance parameter (unknown variable, not a parameter of the model)

Outline

Learning mixture models with imputation

- Mixture of Gaussian parameters are $\{\pi_c, \boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c\}_{c=1}^k$
  - Unfortunately, NLL is non-convex
  - Various optimization methods are used in practice

- If we optimize over $z^{(i)}$, we can decrease NLL with alternating optimization:
  1. Given the clusters $z^{(i)}$, find the most likely parameters
     - Optimize $p(\mathbf{X} \mid \pi, \mu, \Sigma, \mathbf{z})$ in terms of the $\{\pi_c, \boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c\}_{c=1}^k$
     - Set π_c based on frequency of seeing $z^{(i)} = c$
     - Set $\boldsymbol{\mu}_c$ to the mean of examples in cluster c
     - Set $\boldsymbol{\Sigma}_c$ to the covariance of examples in cluster c
  2. Given the parameters, find the most likely clusters
     - For each example i, compute responsibilities $r_c^{(i)} = p(z^{(i)} = c \mid x^{(i)}, \pi_c, \boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)$
     - Set $z^{(i)}$ to the argmax of $r_c^{(i)}$ over c

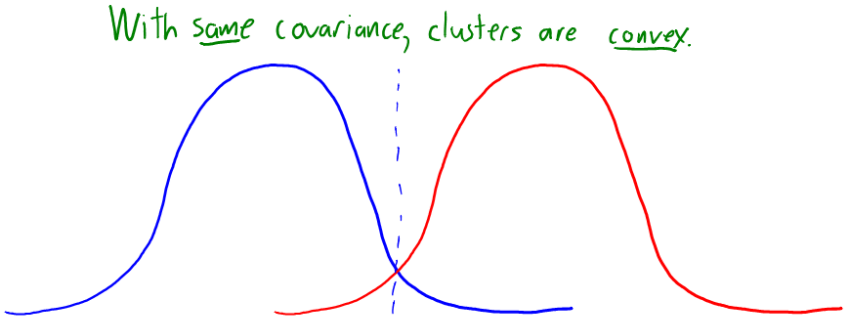
- Connection to Gaussian discriminant analsysis (GDA), using clusters $z^{(i)}$ as labels:
  - Step 1 is the learning step in GDA; Step 2 is the prediction step in GDA

Special Case: k-Means

- Algorithm from the previous slide is a generalization of k-means clustering
- Apply the algorithm assuming $\pi_c = 1/k$ and $\mathbf{\Sigma}_c = \mathbf{I}$ for all c:
  1. Given the clusters z^i, find the most likely parameters
     - Set $\boldsymbol{\mu}_c$ to the mean of examples in cluster c
  2. Given the parameters, find the most likely clusters
     - Set $z^{(i)}$ to the closest mean of example i
- As with k-means, initialization matters for fitting mixture of Gaussians
  - May need to do multiple random restarts, or clever initializations like k-means++

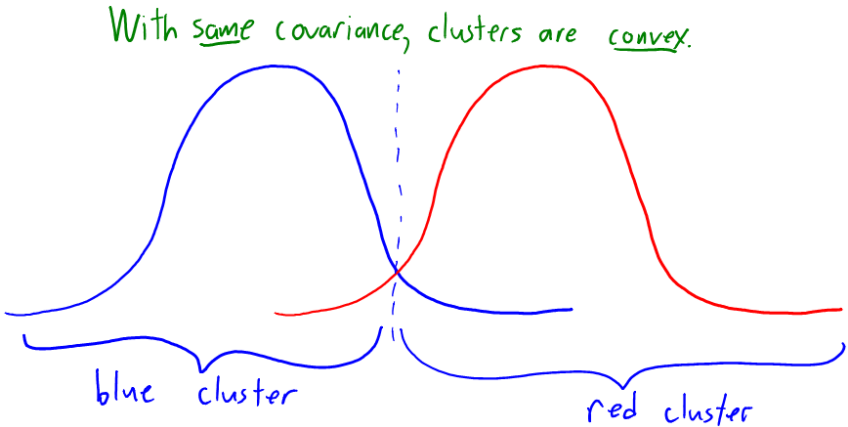
k-Means vs. Mixture of Gaussians

- k-means can be viewed as fitting a Gaussian mixture (all $\pi_c = \frac{1}{k}$, same $\boldsymbol{\Sigma} = \sigma^2\mathbf{I}$)
  - But using a variable $\boldsymbol{\Sigma}_c$ allows non-convex clusters



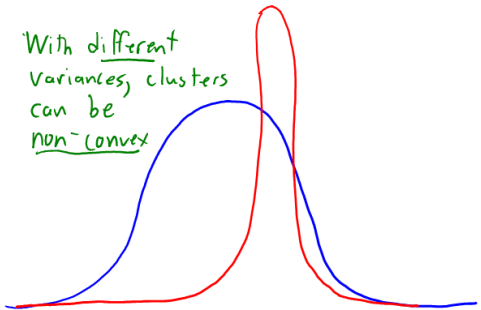
k-Means vs. Mixture of Gaussians

- k-means can be viewed as fitting a Gaussian mixture (all $\pi_c = \frac{1}{k}$, same $\boldsymbol{\Sigma} = \sigma^2 \mathbf{I}$)
  - But using a variable $\boldsymbol{\Sigma}_c$ allows non-convex clusters



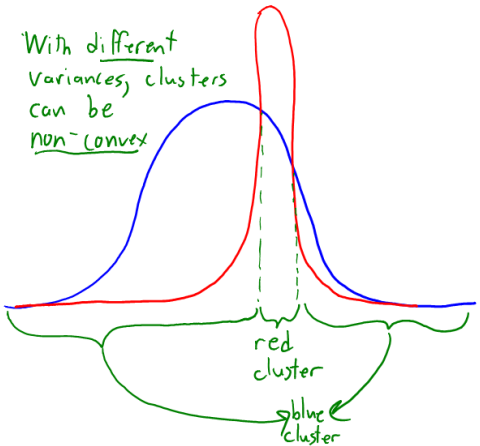
k-Means vs. Mixture of Gaussians

- k-means can be viewed as fitting a Gaussian mixture (all $\pi_c = \frac{1}{k}$, same $\boldsymbol{\Sigma} = \sigma^2\mathbf{I}$)
  - But using a variable $\boldsymbol{\Sigma}_c$ allows non-convex clusters



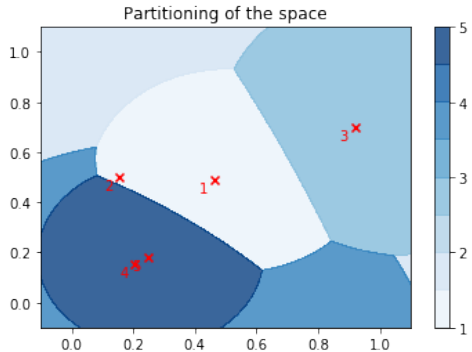
k-Means vs. Mixture of Gaussians

- k-means can be viewed as fitting a Gaussian mixture (all $\pi_c = \frac{1}{k}$, same $\boldsymbol{\Sigma} = \sigma^2\mathbf{I}$)
  - But using a variable $\boldsymbol{\Sigma}_c$ allows non-convex clusters



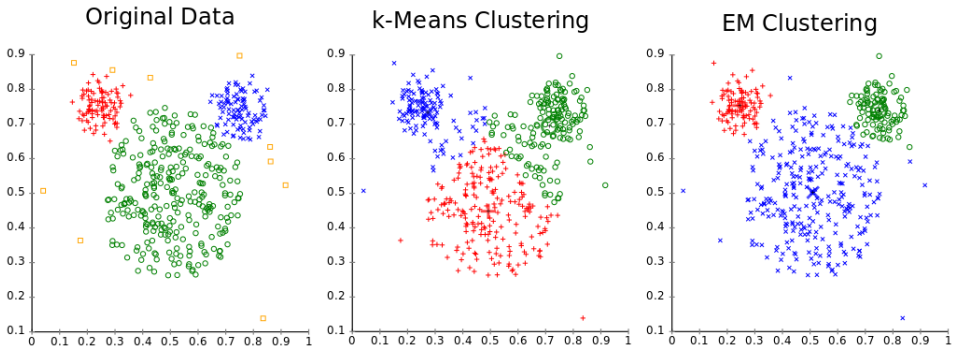
k-Means vs. Mixture of Gaussians

- k-means can be viewed as fitting a Gaussian mixture (all $\pi_c = \frac{1}{k}$, same $\mathbf{\Sigma} = \sigma^2\mathbf{I}$)
  - But using a variable $\mathbf{\Sigma}_c$ allows non-convex clusters



k-Means vs. Mixture of Gaussians

- k-means can be viewed as fitting a Gaussian mixture (all $\pi_c = \frac{1}{k}$, same $\mathbf{\Sigma} = \sigma^2\mathbf{I}$)
  - But using a variable $\mathbf{\Sigma}_c$ allows non-convex clusters



https://en.wikipedia.org/wiki/K-means_clustering

Digression: MLE does not exist

bonus!

- For mixture of at least two Gaussians, there is no MLE
- You can make the likelihood arbitrarily large:
  - Set $\mu_c = x^{(i)}$ for some particular i and c, and make $\boldsymbol{\Sigma}_c \to 0$
  - Optimizers often find models with degenerate components
  - Also often get empty clusters
- It is common to remove empty clusters and use a regularized update,

$$\boldsymbol{\Sigma}_c = \frac{1}{\sum_{i=1}^n r_c^{(i)}} \sum_{i=1}^n r_c^{(i)} (x^{(i)} - \mu_c)(x^{(i)} - \mu_c)^\mathsf{T} + \lambda \mathbf{I}$$

  which is MAP estimation with an L1 regularizer on diagonals of the precision
  - The MAP estimate exists with this and other usual priors on $\boldsymbol{\Sigma}_c$

Outline

Previously: Product of Bernoullis

- A while ago we covered density estimation with discrete variables,

$$\mathbf{X} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

  using a product of Bernoullis:

$$p(x^{(i)} \mid \theta) = \prod_{j=1}^{d} p(x_j^{(i)} \mid \theta_j)$$

- Easy to fit but very strong independence assumption:
  - Knowing $x_j^{(i)}$ tells you nothing about $x_k^{(i)}$

- A more powerful model: mixture of Bernoullis

Mixture of Bernoullis

- Consider a coin flipping scenario where we have two coins:
  - Coin 1 has $\theta_1 = 0.5$ (fair) and coin 2 has $\theta_2 = 1$ (biased)

- Half the time we flip coin 1, and otherwise we flip coin 2:

$$p(x^{(i)} = 1 \mid \theta_1, \theta_2) = \pi_1 \text{ Bern}(x^{(i)} = 1 \mid \theta_1) + \pi_2 \text{ Bern}(x^{(i)} = 1 \mid \theta_2)$$
$$= \frac{1}{2}\theta_1 + \frac{1}{2}\theta_2 = \frac{\theta_1 + \theta_2}{2}$$

- With one variable this mixture model is not very interesting
- It's exactly equivalent to flipping one coin with $\theta = 0.75$

- But mixture of product of Bernoullis can model dependencies...

Mixture of Independent Bernoullis

- Consider a mixture of a product of Bernoullis:

$$p(x \mid \theta_1, \theta_2) = \underbrace{\frac{1}{2}\prod_{j=1}^{d} \text{Bern}(x_j \mid \theta_{j|1})}_{\text{first set of Bernoullis}} + \underbrace{\frac{1}{2}\prod_{j=1}^{d} \text{Bern}(x_j \mid \theta_{j|2})}_{\text{second set of Bernoullis}}$$

- Conceptually, we now have two sets of coins:
  - Half the time we throw the first set, half the time we throw the second set

- With $d = 4$ we could have $\theta_{\cdot|1} = \begin{bmatrix} 0 & 0.7 & 1 & 1 \end{bmatrix}$ and $\theta_{\cdot|2} = \begin{bmatrix} 1 & 0.7 & 0.8 & 0 \end{bmatrix}$
  - Half the time we have $p(x_3^{(i)} = 1) = 1$, half the time it's 0.8

- Have we gained anything?

Mixture of Independent Bernoullis

- Previous example: $\theta_{\cdot|1} = \begin{bmatrix} 0 & 0.7 & 1 & 1 \end{bmatrix}$ and $\theta_{\cdot|2} = \begin{bmatrix} 1 & 0.7 & 0.8 & 0 \end{bmatrix}$
- Here are some samples from this model:

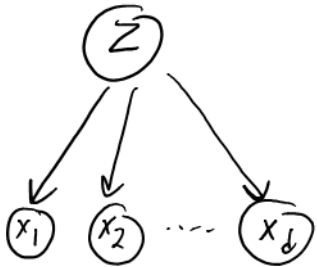
$$\mathbf{X} = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}$$

- Unlike product of Bernoullis, features in samples are not independent
  - In this example knowing $x_1 = 1$ tells you that $x_4 = 0$

- This model can capture dependencies: $\underbrace{p(x_4 = 1 \mid x_1 = 1)}_{0} \neq \underbrace{p(x_4 = 1)}_{0.5}$

Mixture of Independent Bernoullis

- Drawing the mixture of Bernoullis as a directed acyclic graph (DAG):



- If we know z, then each x_j is independent
- Since we usually don't, there are dependencies between the x_j
  - We'll talk a bunch about this kind of reasoning soon ("graphical models")

- This is the same graph as naive Bayes, with cluster z instead of class y
  - If you see one spammy word, it makes other spammy words more likely

Mixture of Independent Bernoullis

- General mixture of independent Bernoullis:

$$p(x \mid \Theta) = \sum_{c=1}^{k} \pi_c \, p(x \mid z = c) = \sum_{c=1}^{k} \left[\pi_c \prod_{j=1}^{d} \theta_{j|c} \right]$$

  - Here Θ contains all the parameters: k values of π_c, and $k \times d$ values of $\theta_{j|c}$

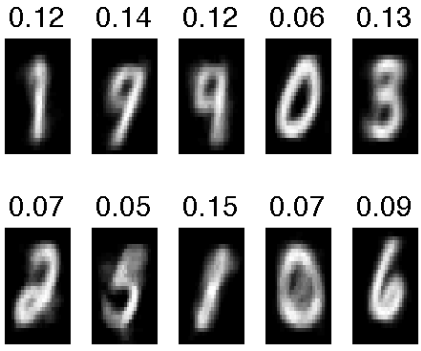
- Mixture of Bernoullis can model dependencies between variables
  - Individual mixtures act like clusters of the binary data
  - Knowing cluster of one variable gives information about other variables

- With k large enough, mixture of Bernoullis can model any binary distribution
  - With $k = 2^d$, we can make all the $\theta_{j|c} \in \{0, 1\}$, and it becomes a tabular distribution
  - Hopefully, we can make a useful model with $k \ll 2^d \ldots$

Mixture of Independent Bernoullis

- Plotting parameters θ_c with 10 mixtures trained on MNIST digits (with "EM"):

(numbers above images are mixture coefficients π_c)

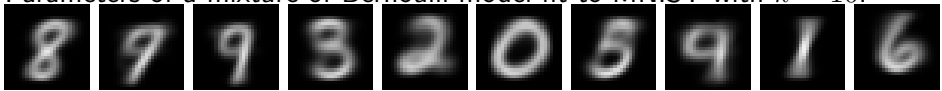


http://pmtk3.googlecode.com/svn/trunk/docs/demoOutput/bookDemos/%2811%29-Mixture_models_and_the_EM_algorithm/mixBerMnistEM.html

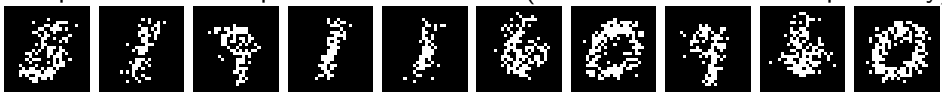
- Remember this is unsupervised: it hasn't been told there are ten digits.
  - You could use this model to "fill in" missing parts of an image.

Mixture of Bernoullis on Digits with $k > 10$

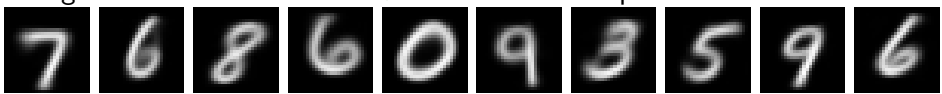
- Parameters of a mixture of Bernoulli model fit to MNIST with $k = 10$:



- Samples better than product of Bernoullis (but no within-cluster dependency):



- You get a better model with $k > 10$. First 10 components with $k = 50$:



- Samples from the $k = 50$ model (can have more than one "type" of a number):

Outline

Big Picture: Training and Inference

- Many possible mixture model inference tasks:
  - Generate samples
  - Measure likelihood of test examples $\tilde{x}$
    - To detect outliers, for example
  - Compute probability that test example belongs to cluster c
  - Compute marginal or conditional probabilities
  - "Fill in" missing parts of a test example

- Mixture model training phase:
  - Input is a matrix $\mathbf{X}$, number of clusters k, and form of individual distributions
  - Output is mixture proportions π_c and parameters of components
    - The $\theta_{\cdot|c}$ for Bernoulli, and the $\{\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c\}$ for Gaussians
    - Also, maybe, the responsibilities $r_c^{(i)}$ or cluster assignments $z^{(i)}$

Fitting a Mixture of Bernoullis: Imputation of $z^{(i)}$

- Imputation approach to fitting mixture of Bernoullis, optimizing the $z^{(i)}$:
  1. Find the most likely cluster $z^{(i)}$ for each example $x^{(i)}$,

$$z^{(i)} \in \arg\max_c p(z^{(i)} = c \mid x^{(i)}, \Theta)$$

  2. Update the mixture probabilities as proportion of examples in cluster,

$$\pi_c = \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}(z^{(i)} = c)$$

  3. Update the product of Bernoullis based on examples in cluster,

$$\theta_{j|c} = \frac{\sum_{i=1}^{n} \mathbb{1}(z^{(i)} = c) x_j^{(i)}}{\sum_{i=1}^{n} \mathbb{1}(z^{(i)} = c)}$$

- This picks a particular value for each $z^{(i)}$; sometimes called "hard assignments"

Fitting a Mixture of Bernoullis: Expectation Maximization

- Expectation maximization (EM) approach to fitting mixture of Bernoulli:
  1. Find the responsibility of cluster $z^{(i)}$ for each example $x^{(i)}$

$$r_c^{(i)} = p(z^{(i)} = c \mid x^{(i)}, \Theta)$$

  2. Update the mixture probabilities as proportion of examples cluster is responsible for,

$$\pi_c = \frac{1}{n} \sum_{i=1}^{n} r_c^{(i)}$$

  3. Update the product of Bernoullis based on examples cluster is responsible for,

$$\theta_{j|c} = \frac{\sum_{i=1}^{n} r_c^{(i)} x_j^{(i)}}{\sum_{i=1}^{n} r_c^{(i)}}$$

- This does "soft" (probabilistic) assignment for the $z^{(i)}$ variables

Fitting a Mixture of Gaussians: Expectation Maximization

- Expectation maximization (EM) approach to ftting mixture of Gaussians:
  1. Find the responsibility of cluster $z^{(i)}$ for each example $x^{(i)}$

$$r_c^{(i)} = p(z^{(i)} = c \mid x^{(i)}, \Theta)$$

  2. Update the mixture probabilities as proportion of examples cluster is responsible for,

$$\pi_c = \frac{1}{n} \sum_{i=1}^{n} r_c^{(i)}$$

  3. Update the Gaussian based on examples cluster is responsible for,

$$\boldsymbol{\mu}_c = \frac{1}{\sum_{i=1}^{n} r_c^{(i)}} \sum_{i=1}^{n} r_c^{(i)} x^{(i)}, \quad \boldsymbol{\Sigma}_c = \frac{1}{\sum_{i=1}^{n} r_c^{(i)}} \sum_{i=1}^{n} r_c^{(i)} \left(x^{(i)} - \mu_c\right)\left(x^{(i)} - \mu_c\right)^{\mathsf{T}}$$

- Video: `https://www.youtube.com/watch?v=B36fzChfyGU`

Expectation Maximization vs. Imputation

- The imputation method is optimizing $p(x^{(i)}, z^{(i)} \mid \Theta)$ in terms of $z^{(i)}$ and Θ
- So we're optimizing $z^{(i)}$ as well as Θ
  - $p(x^{(i)}, z^{(i)} \mid \Theta)$ is called the complete-data likelihood

- Expectation maximization (EM) is optimizing $p(x^{(i)} \mid \Theta)$ in terms of Θ
  - So we're integrating over $z^{(i)}$ values while optimizing Θ
    - $p(x^{(i)} \mid \Theta)$ is the usual likelihood, marginalizing over the $z^{(i)}$

- EM is a general algorithm for parameter learning with missing data
  - For mixtures, the "missing" data is the $z^{(i)}$ variables
  - But EM can be used for any probabilistic model where we have missing data

Expectation Maximization: General Form

- With data $\mathbf{X}$ and hidden values $\mathbf{Z}$, general EM uses iterations of the form

$$\Theta_{t+1} \in \arg\max_{\Theta} \sum_{\mathbf{Z}} p(\mathbf{Z} \mid \mathbf{X}, \Theta_t) \log p(\mathbf{X}, \mathbf{Z} \mid \Theta)$$
$$= \arg\max_{\Theta} \mathop{\mathbb{E}}_{\mathbf{Z}|\mathbf{X},\Theta_t} [\log p(\mathbf{X}, \mathbf{Z} \mid \Theta)]$$

- Summing/integrating over all possible hidden values $\mathbf{Z}$ may be hard
  - But in many cases this simplifies, due to conditional independence assumptions

- For mixture models, the EM iteration simplifies to (see notes on webpage)

$$\sum_{i=1}^{n} \sum_{z^{(i)}=1}^{k} \underbrace{p(z^{(i)} \mid x^{(i)}, \Theta_t)}_{\text{responsibility}} \underbrace{\log p(x^{(i)}, z^{(i)} \mid \Theta)}_{\text{complete-data log-lik}}$$

so summing over k^n possible clusterings turns into sum over nk terms

"E-Step" and "M-Step" for Mixture Models

- For mixture models, EM is often written as two steps:
  1. E-step: compute responsibilities $r_c^{(i)}$ for all i and c, for current Θ_t
  2. M-step: optimize the weighted "complete-data" log-likelihood

$$\Theta_{t+1} \in \arg\max_{\Theta} \sum_{i=1}^{n} \sum_{z^{(i)}=1}^{k} r_c^{(i)} \log p(x^{(i)}, z^{(i)} \mid \Theta)$$

- For other models, there may not be separate "E-steps" and "M-steps"
- EM is most useful when complete-data log-likelihood is easy to optimize
- Most common case: complete-data log-likelihood is in an exponential family
  - Mixture of Bernoullis, mixture of Gaussians, etc ewc
  - Here the M-step is a weighted combination of the sufficient statistics

Expectation Maximization Algorithm: Properties

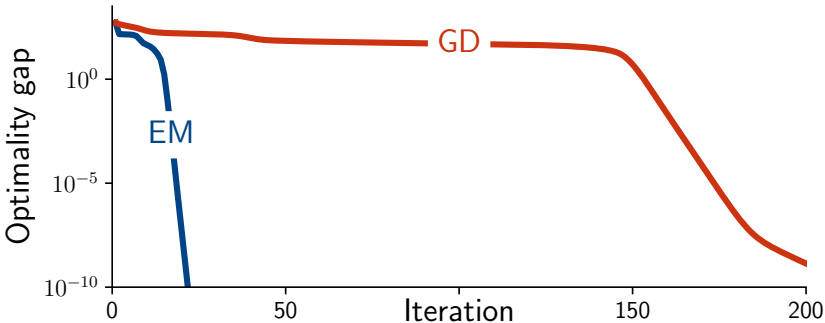
- EM monotonically increases likelihood, $p(\mathbf{X} \mid \Theta_{t+1}) \geq p(X \mid \Theta_t)$
  - Useful for debugging: if likelihood decreases, you have a bug
- EM doesn't need a step size, unlike many learning algorithms
- EM tends to satisfy constraints automatically
  - Unlike gradient descent, don't need to worry about constraints on π_c and Σ_c
    - Assuming you have a prior to avoid degenerate situations where MLE does not exist
- EM iterations are parameterization-independent
  - Get the same performance under any re-parameterization of the problem
- EM is notorious for converging to bad local optima
  - Not really the algorithm's fault: we typically apply EM to hard problems

Expectation Maximization Algorithm: Properties

- EM converges to a stationary point, under weak assumptions
- EM is at least as fast as gradient descent (with a constant step size)
  - In the worst case, for differentiable problems
  - EM can also be used for non-differentiable likelihoods
- EM converges faster as entropy of hidden variables decreases
  - If value of hidden variables is "obvious", it converges very fast
- EM can be arbitrarily faster than gradient descent
- Mark has a bunch of more detailed material on the EM algorithm here:
  - `https://www.cs.ubc.ca/~schmidtm/Courses/440-W22/L34.5.pdf`

Expectation Maximization vs. Gradient Descent

- Expectation maximization vs. gradient for fitting mixture of 2 Gaussians:



Outline

Combining Mixture Models with Other Models

bonus!

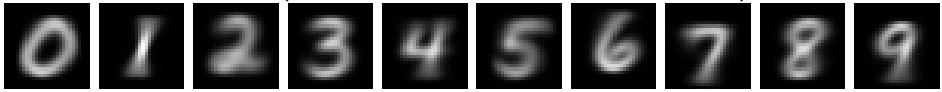
- We can use mixtures in generative classifiers
  - Model $p(x \mid y)$ as a mixture instead of simple Gaussian or product of Bernoullis
    - VQNB from Assignment 2 fits a mixture of Bernoullis for each class

- We can do mixture of more-complicated distributions:
  - Mixture of categoricals (can model arbitrary categorical vectors)
  - Mixture of student-t distributions
    - Not exponential family, so no simple closed-form update of parameters
  - Mixture of Markov chains, DAGs/UGMs (next topics in course)

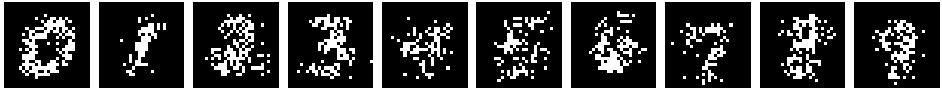
- We can add features to mixture models for supervised learning:
  - Mixture of experts: have k regression/classification models
    - Each model can be viewed as a "expert" for a cluster of $x^{(i)}$ values
    - GPT-4, Grok, ... are mixtures of Transformers
    - These models use conditional weights π_c; some are 0 for computational savings

Less-Naive Bayes on Digits

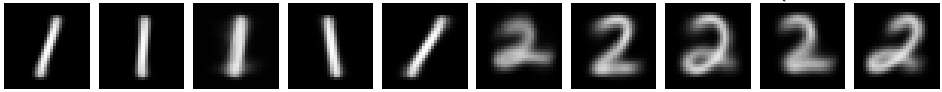
- Naive Bayes θ_c values (independent Bernoullis for each class):



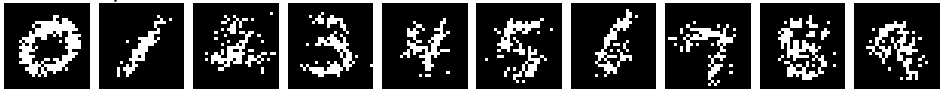
- One sample from each class:



- Generative classifier with mixture of 5 Bernoullis for each class (digits 1 and 2):



- One sample from each class:



- Would get less noisy samples and more variation with mixture of graphical models

Dirichlet Process

bonus!

- Non-parametric Bayesian methods allow us to consider infinite mixture model,

$$p(x \mid \Theta) = \sum_{c=1}^{\infty} \pi_c \, p_c(x \mid \Theta_c)$$

- Common choice for prior on π values is Dirichlet process:
  - Also called "Chinese restaurant process" and "stick-breaking process"
  - For finite datasets, only a fixed number of clusters have $\pi_c \neq 0$
  - But don't need to pick number of clusters; it grows with data size

- Gibbs sampling in Dirichlet process mixture model in action:
  `https://www.youtube.com/watch?v=0Vh7qZY9sPs`

Dirichlet Process

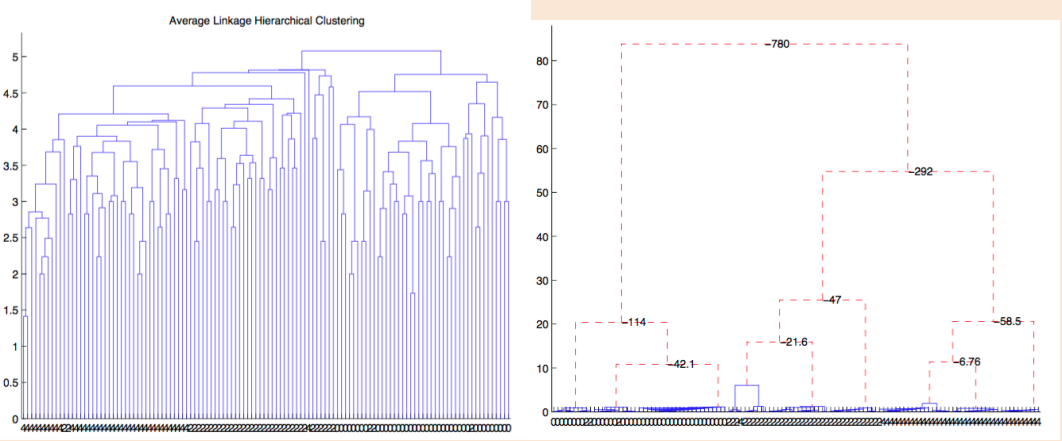
bonus!

- Slides giving more details on Dirichelt process mixture models:
  - `https://www.cs.ubc.ca/labs/lci/mlrg/slides/NP.pdf`
- We could alternately put a prior on number of clusters k:
  - Allows more flexibility than Dirichlet process as a prior
  - Needs "trans-dimensional" MCMC to sample models of different sizes
- There are a variety of interesting variations on Dirichlet processes
  - Beta process ("Indian buffet process")
  - Hierarchical Dirichlet process
  - Polya trees
  - Infinite hidden Markov models

Bayesian Hierarchical Clustering

bonus!

- Hierarchical clustering of $\{0, 2, 4\}$ digits using classic and Bayesian method:

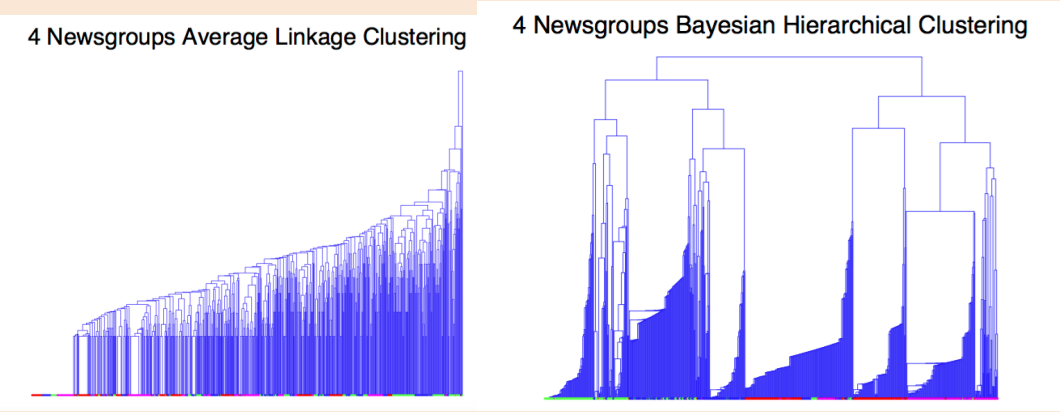


http://www2.stat.duke.edu/~kheller/bhcnew.pdf (y-axis represents distance between clusters)

Bayesian Hierarchical Clustering

bonus!

- Hierarchical clustering of newgroups using classic and Bayesian method:



http://www2.stat.duke.edu/~kheller/bhcnew.pdf (y-axis represents distance between clusters)

Continuous Mixture Models

bonus!

- We can also consider mixture models where $z^{(i)}$ is continuous,

$$p(x^{(i)}) = \int_{z^{(i)}} p(z^{(i)})p(x^{(i)} \mid z^{(i)} = c)\mathrm{d}z^{(i)}$$

- Unfortunately, computing the integral might be hard

- Special case is if both probabilities are Gaussian (conjugate)
  - Leads to probabilistic PCA and factor analysis (OCEAN model in psychology)
  - Mark's old material:
    `https://www.cs.ubc.ca/~schmidtm/Courses/540-W19/L17.5.pdf`
- Another special case is scale mixtures of Gaussians
  - $p(x^{(i)} \mid z^{(i)})$ is Gaussian, and $p(z^{(i)})$ is a gamma prior on variance (conjugate)
  - Can represent many distributions in this form, like Laplace and student-t
  - Leads to EM algorithms for fitting Laplace and student-t

Outline

Non-Parametric Mixtures: Kernel Density Estimation

- A common non-parametric mixture model centers one cluster on each example:

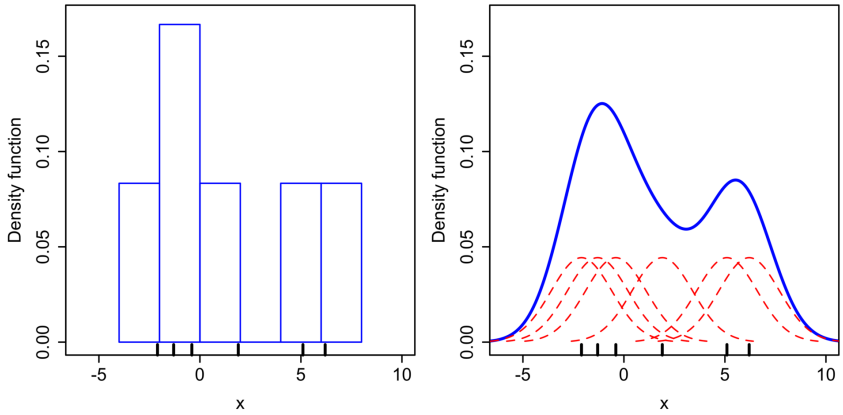
$$p(x^{(i)}) = \frac{1}{n} \sum_{j=1}^{n} \mathcal{N}(x^{(i)} \mid x^{(j)}, \sigma^2 \mathbf{I})$$

- This is called kernel density estimation (KDE) or the Parzen window method
  - Don't have to use a normal likelihood, though that's a common choice
  - Scale σ^2 is viewed as a hyper-parameter

- By fixing mean/covariance/k, σ^2 is the only parameter: otherwise immediate from $\mathbf{X}$
  - Most inference tasks (except finding the mode) are easy, but slow (depend on n)
  - Many variations exist; see bonus slides for generalizations
    - Tends to work great in low dimensions, and poorly in high dimensions

Histogram vs. Kernel Density Estimator

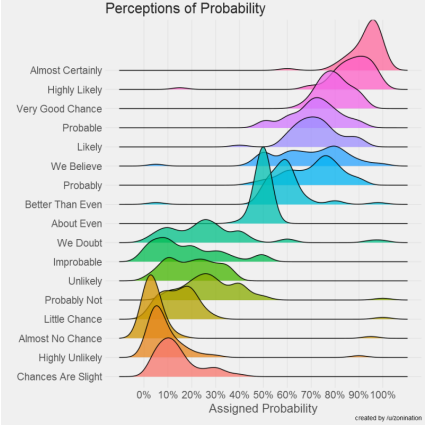
- You can think of a kernel density estimate as a continuous histogram:



https://en.wikipedia.org/wiki/Kernel_density_estimation

Kernel Density Estimator for Visualization

- Visualization of people's opinions about what "likely" and other words mean.

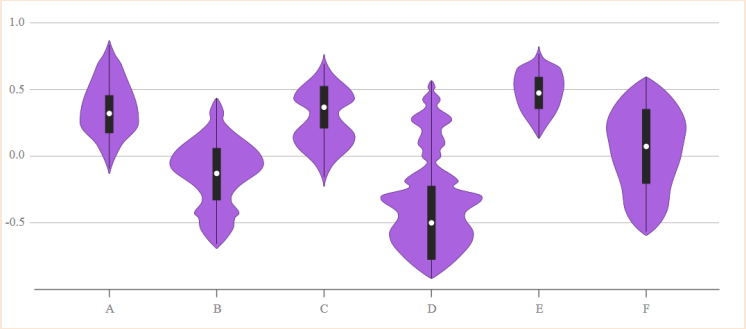


http://blog.revolutionanalytics.com/2017/08/probably-more-probably-than-probable.html

Violin Plot: Added KDE to a Boxplot

bonus!

- Violin plot adds KDE to a boxplot:

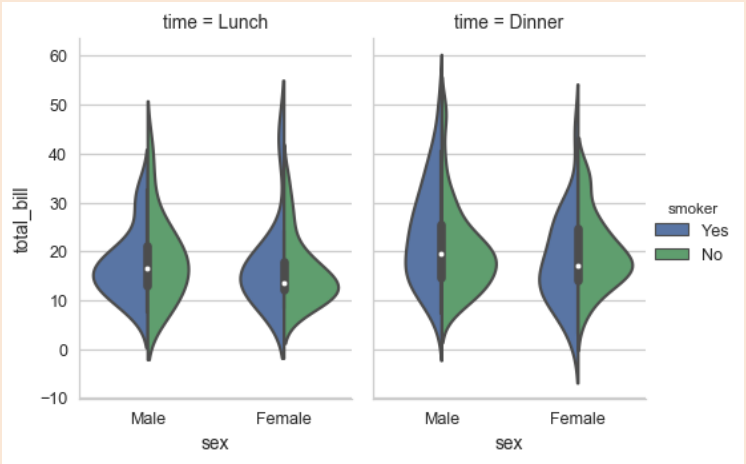


https://datavizcatalogue.com/methods/violin_plot.html

Violin Plot: Added KDE to a Boxplot

bonus!

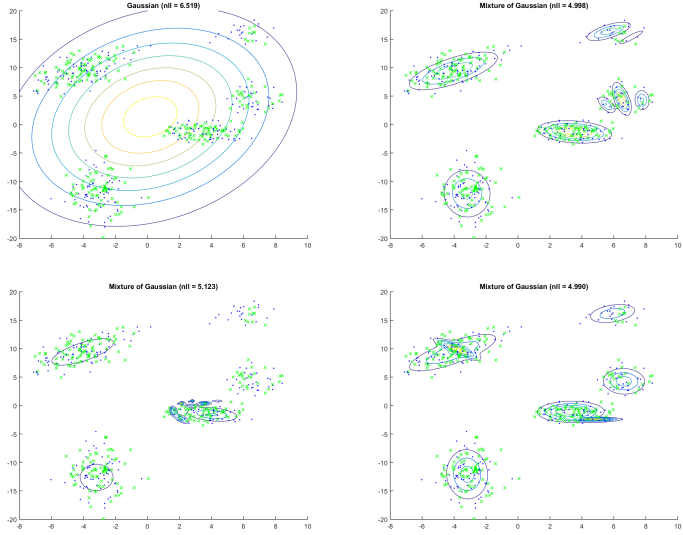
- Violin plot adds KDE to a boxplot:



https://seaborn.pydata.org/generated/seaborn.violinplot.html

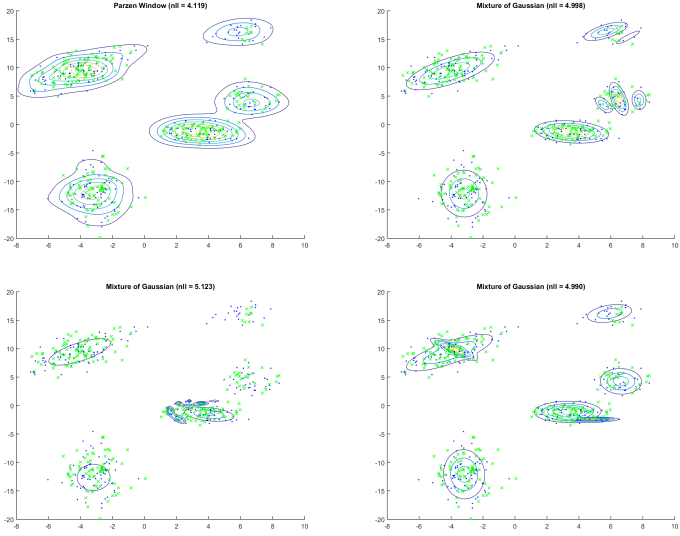
KDE vs. Mixture of Gaussian

- Multivariate vs mixture of Gaussians (different EM initializations):



KDE vs. Mixture of Gaussian

- Kernel density estimation vs mixture of Gaussians (different EM initializations):



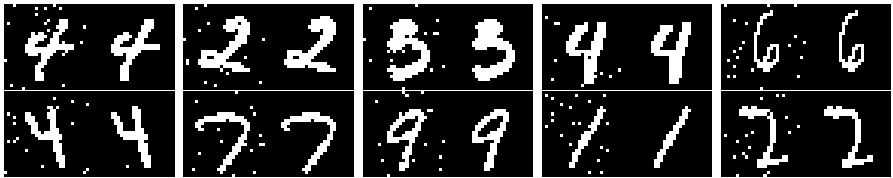
Mean-Shift Clustering

bonus!

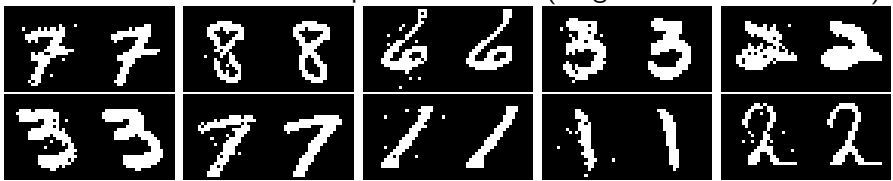
- Mean-shift clustering uses KDE for clustering:
  - Define a KDE on the training examples, and then for test example $\hat{x}$:
    - Run gradient descent to maximize $p(x)$ starting from $\hat{x}$
  - Clusters are points that reach same local minimum
- `https://spin.atomicobject.com/2015/05/26/mean-shift-clustering`
- Not sensitive to initialization, no need to choose k, can find non-convex clusters
- Similar to density-based clustering from 340
  - Doesn't require uniform density within cluster
  - Can be used for vector quantization
- "The 5 Clustering Algorithms Data Scientists Need to Know":
  - `https://towardsdatascience.com/the-5-clustering-algorithms-data-scientists-need-to-know-a36d136ef68`

Kernel Density Estimation on Digits

- Samples from a KDE model of digits:
  - Sample is on the left, right is the closest image from the training set.



- KDE just samples a training example then adds noise
  - Usually makes more sense for continuous data that is densely packed
- A variation with a location-specific variance (diagonal Σ instead of $\sigma^2\mathbf{I}$):



Summary

- Mixture of Gaussians writes probability as convex combo of Gaussian densities
  - Can model arbitrary continuous densities
- Latent-variable representation of mixutres with cluster variables $z^{(i)}$
  - Allows ancestral sampling by sampling cluster than example
  - Responsibility is probability that an example belongs to a cluster
- Mixture of Bernoullis can model dependencies between discrete variables
  - Unsupervised version of naive Bayes; can model arbitrary binary distributions
- Learning by alternating imputing z^i and fitting full model. . . or more commonly,
- Expectation maximization: algorithm for optimization with hidden variables
  - Instead of imputation, works with "soft" assignments to nuisance variables
  - Maximizes log-likelihood, weighted by all imputations of hidden variables
  - Simple and intuitive updates for fitting mixtures models
  - Appealing properties as an optimization algorithm, but only finds local optimum
- Kernel density estimation: non-parametric density estimation method
  - Center a mixture on each datapoint (smooth variation on histograms)
  - Data visualization, low-dimensional density estimation, mean-shift clustering
- Next time: Markov chains

Avoiding Underflow when Computing Responsibilities

bonus!

- Computing responsibility may underflow for high-dimensional $x^{(i)}$, due to $p(x^{(i)} \mid z^{(i)} = c, \Theta^t)$
- Usual ML solution: do all but last step in log-domain

$$\log r_c^i = \log p(x^i \mid z^i = c, \Theta^t) + \log p(z^i = c \mid \Theta^t) - \log \left(\sum_{c'=1}^{k} p(x^i \mid z^i = c', \Theta^t) p(z^i = c' \mid \Theta^t) \right).$$

- To compute last term, use "log-sum-exp" trick
  - `scipy.special.logsumexp`

Log-Sum-Exp Trick

bonus!

- To compute $\log(\sum_i \exp(v_i))$, set $\beta = \max_i\{v_i\}$ and use:

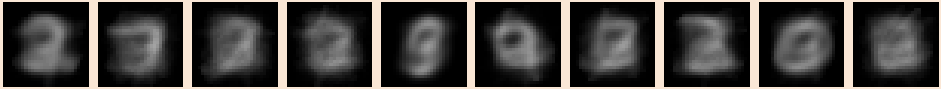
$$\begin{aligned}
\log(\sum_c \exp(v_i)) &= \log(\sum_i \exp(v_i - \beta + \beta)) \\
&= \log(\sum_i \exp(v_i - \beta)\exp(\beta)) \\
&= \log(\exp(\beta)) \sum_i \exp(v_i - \beta)) \\
&= \log(\exp(\beta)) + \log(\sum_i \exp(v_i - \beta)) \\
&= \beta + \log(\sum_i \underbrace{\exp(v_i - \beta)}_{\leq 1}).
\end{aligned}$$

- Avoids overflows in to computing $\exp$ operator

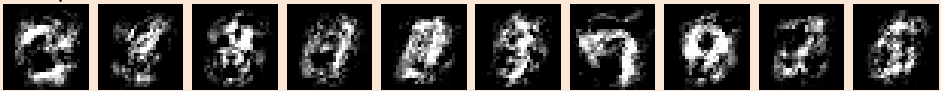
Mixture of Gaussians on Digits

bonus!

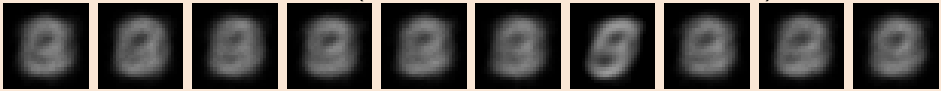
- Mean parameters of a mixture of Gaussians with $k = 10$:



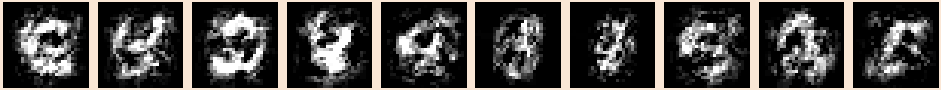
- Samples:



- 10 components with $k = 50$ (might need a better initialization):



- Samples:



EM for MAP Estimation

bonus!

- We can also use EM for MAP estimation. With a prior on Θ our objective is:

$$\underbrace{\log p(X \mid \Theta) + \log p(\Theta)}_{\text{what we optimize in MAP}} = \log \left(\sum_Z p(X, Z \mid \Theta) \right) + \log p(\Theta).$$

- EM iterations take the form of a regularized weighted "complete" NLL,

$$\Theta^{t+1} \in \arg\max_{\Theta} \left\{ \underbrace{\sum_Z p(Z \mid X, \Theta^t) \log p(X, Z \mid \Theta)}_{} + \log p(\Theta) \right\},$$

- Now guarantees monotonic improvement in MAP objective.
  - Has a closed-form solution for mixture of exponential families with conjugate priors.

- For mixture of Gaussians with $-\log p(\Theta_c) = \lambda \text{Tr}(\Theta_c)$ for precision matrices Θ_c:
  - Closed-form solution that satisfies positive-definite constraint (no $\log|\Theta|$ needed).

Generative Mixture Models and Mixture of Experts

bonus!

- Classic generative model for supervised learning uses

$$p(y^i \mid x^i) \propto p(x^i \mid y^i)p(y^i),$$

  and typically $p(x^i \mid y^i)$ is assumed Gaussian (LDA) or independent (naive Bayes).
- But we could allow more flexibility by using a mixture model,

$$p(x^i \mid y^i) = \sum_{c=1}^{k} p(z^i = c \mid y^i)p(x^i \mid z^i = c, y^i).$$

- Another variation is a mixture of disciminative models (like logistic regression),

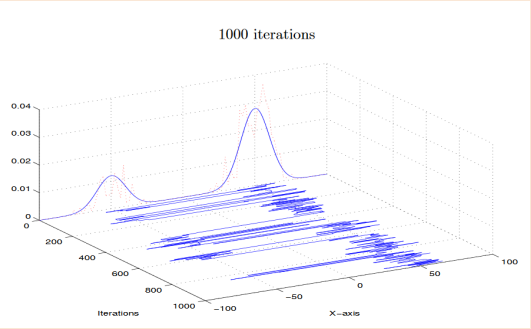
$$p(y^i \mid x^i) = \sum_{c=1}^{k} p(z^i = c \mid x^i)p(y^i \mid z^i = c, x^i).$$

- Called a "mixture of experts" model:
  - Each regression model becomes an "expert" for certain values of x^i.

Mixtures as Proposals in Metropolis-Hastings

bonus!

- Suppose we want to sample from a multi-modal distribution:



http://www.cs.ubc.ca/~arnaud/stat535/slides10.pdf

- With random walk proposals, we stay in one mode for a long time.
- We could instead use mixture model as a proposal in Metropolis-Hastings.
  - Proposal could be a mixture between random walk and "mode jumping".

General Kernel Density Estimation

bonus!

- The 1D kernel density estimation (KDE) model uses

$$p(x^i) = \frac{1}{n}\sum_{j=1}^{n} k_\sigma(\underbrace{x^i - x^j}_{r}),$$

  where the PDF k is called the "kernel" and parameter σ is the "bandwidth".
- In the previous slide we used the (normalized) Gaussian kernel,

$$k_1(r) = \frac{1}{\sqrt{2\pi}}\exp\left(-\frac{r^2}{2}\right), \quad k_\sigma(r) = \frac{1}{\sigma\sqrt{2\pi}}\exp\left(-\frac{r^2}{2\sigma^2}\right).$$

- Note that we can add a "bandwith" (standard deviation) σ to any PDF k_1, using

$$k_\sigma(r) = \frac{1}{\sigma}k_1\left(\frac{r}{\sigma}\right),$$

  from the change of variables formula for probabilities ($|\frac{d}{dr}\left[\frac{r}{\sigma}\right]| = \frac{1}{\sigma}$).
- Under common choices of kernels, KDEs can model any continuous density.

Efficient Kernel Density Estimation

bonus!

- KDE with the Gaussian kernel is slow at test time:
  - We need to compute distance of test point to every training point.
- A common alternative is the Epanechnikov kernel,

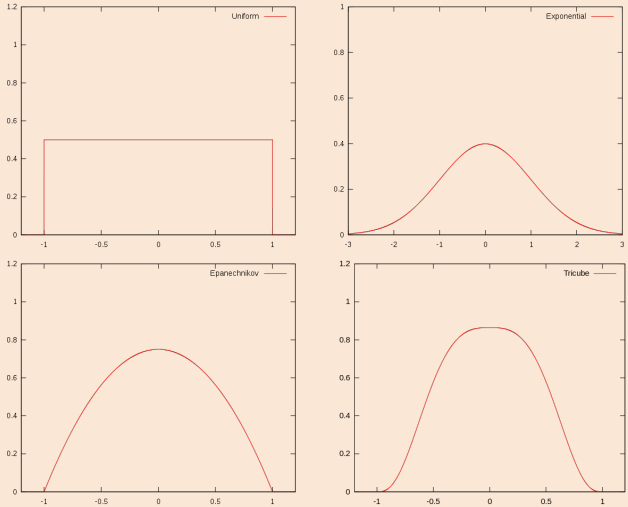
$$k_1(r) = \frac{3}{4}\left(1 - r^2\right)\mathcal{I}\left[|r| \leq 1\right].$$

- This kernel has two nice properties:
  - Epanechnikov showed that it is asymptotically optimal in terms of squared error.
  - It can be much faster to use since it only depends on nearby points.
    - You can use hashing to quickly find neighbours in training data.
- It is non-smooth at the boundaries but many smooth approximations exist.
  - Quartic, triweight, tricube, cosine, etc.
- For low-dimensional spaces, we can also use the fast multipole method.

Visualization of Common Kernel Functions

bonus!

Histogram vs. Gaussian vs. Epanechnikov vs. tricube:



https://en.wikipedia.org/wiki/Kernel_%28statistics%29

Multivariate Kernel Density Estimation

bonus!

- The multivariate kernel density estimation (KDE) model uses

$$p(x^i) = \frac{1}{n}\sum_{j=1}^{n} k_A(\underbrace{x^i - x^j}_{r}),$$

- The most common kernel is a product of independent Gaussians,

$$k_I(r) = \frac{1}{(2\pi)^{\frac{d}{2}}} \exp\left(-\frac{\|r\|^2}{2}\right).$$

- We can add a bandwith matrix A to any kernel using

$$k_A(r) = \frac{1}{|A|} k_1(A^{-1}r) \qquad \text{(generalizes } k_\sigma(r) = \frac{1}{\sigma} k_1\left(\frac{r}{\sigma}\right)\text{)},$$

  and in Gaussian case we get a multivariate Gaussian with $\Sigma = AA^T$.

- To reduce number of parameters, we typically:
  - Use a product of independent distributions and use $A = \sigma I$ for some σ.