Mixture distributions

CPSC 440/550: Advanced Machine Learning

`cs.ubc.ca/~dsuth/440/24w2`

University of British Columbia, on unceded Musqueam land

2024-25 Winter Term 2 (Jan–Apr 2025)

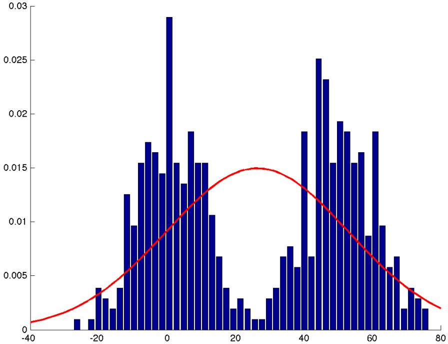
Last time: Exponential families

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

Outline

1 Gaussian for Multi-Modal Data

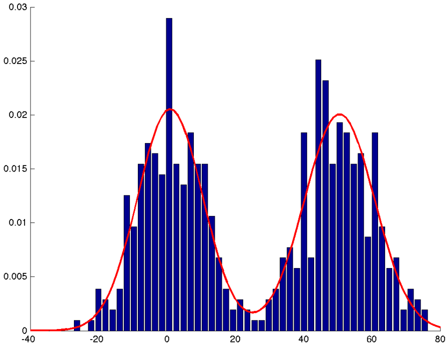
- One major drawback of Gaussian is that it is uni-modal
  - It gives a terrible fit to data like this:



- 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



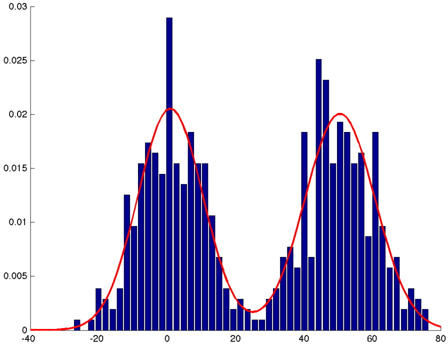
- Half the samples are from Gaussian one, half are from Gaussian two

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

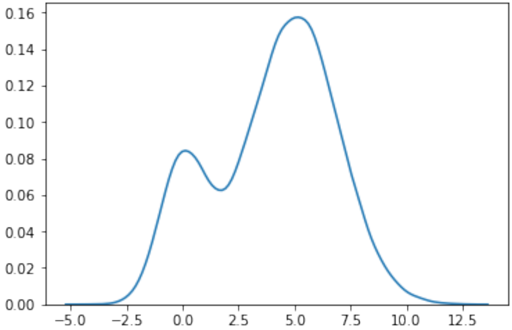


Mixture of Gaussians

- 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"

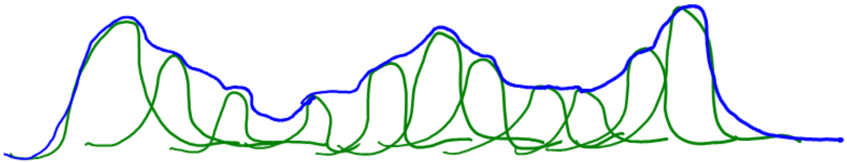


Mixture of Gaussians

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

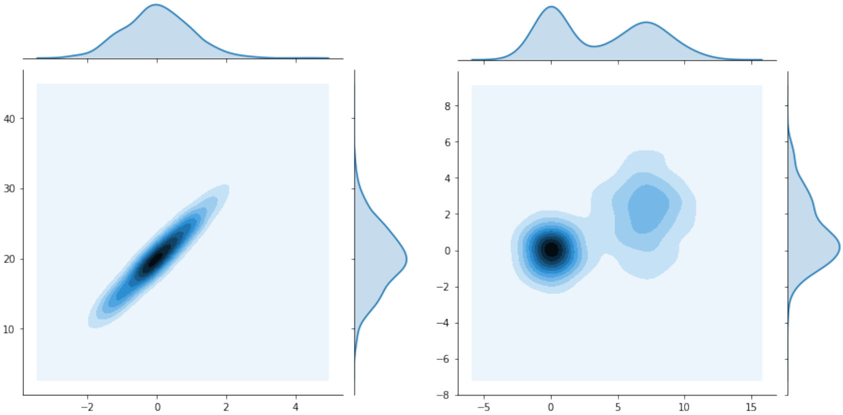
$$p(x \mid \mu, \Sigma, \pi) = \sum_{c=1}^{k} \pi_c \underbrace{\mathcal{N}(x \mid \boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)}_{\text{pdf of Gaussian } c}$$

- π_c are categorical distribution parameters (non-negative, 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 bounded subset of $\mathbb{R}^d$



Mixture of Gaussians

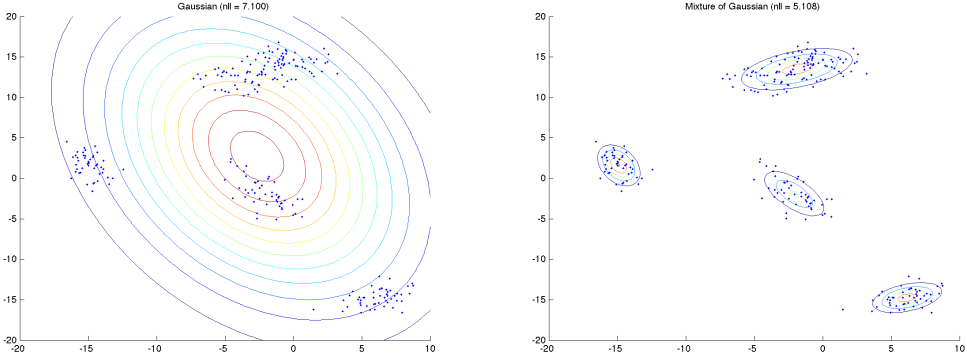
- Gaussian versus mixture of two Gaussians in 2D:



- Marginals will also be mixtures of Gaussians

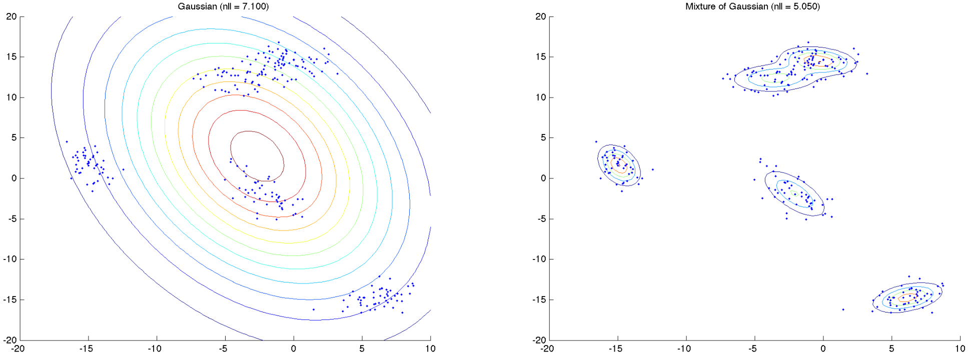
Mixture of Gaussians

- Gaussian versus mixture of four Gaussians for 2D multi-modal data:



Mixture of Gaussians

- Gaussian vs. mixture of five 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 $\{1, 2, \ldots, k\}$ when we have k mixtures
  - The value $z^{(i)}$ represents "which component this example came from"
  - We do not observe the $z^{(i)}$ values (called latent variables)

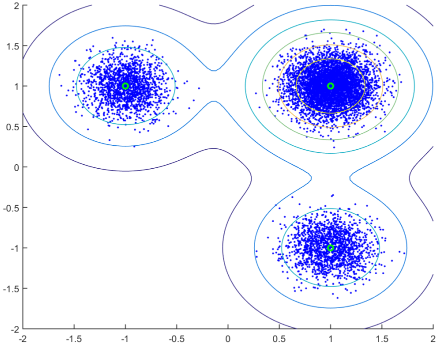
- Why this interpretation of "each $x^{(i)}$ comes from one Gaussian"?
  - Consider a model where $p(Z = c) = \pi_c$, and $X \mid (Z = c) \sim \mathcal{N}(\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)$
  - Now marginalize over the $z^{(i)}$ in this model:

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

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

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:

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

  - 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 part 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
     - That is, optimize $p(\mathbf{X} \mid \pi, \mu, \Sigma, \mathbf{z})$ with respect to $\{\pi_c, \boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c\}_{c=1}^k$, for frozen $(z^{(i)})$
     - 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 $r_c^{(i)} = p(z^{(i)} = c \mid x^{(i)}, \pi, \mu, \Sigma)$
     - Set $z^{(i)} = \arg\max_c r_c^{(i)}$

- Connection to Gaussian discriminant analsysis (GDA), using clusters $z^{(i)}$ as labels:

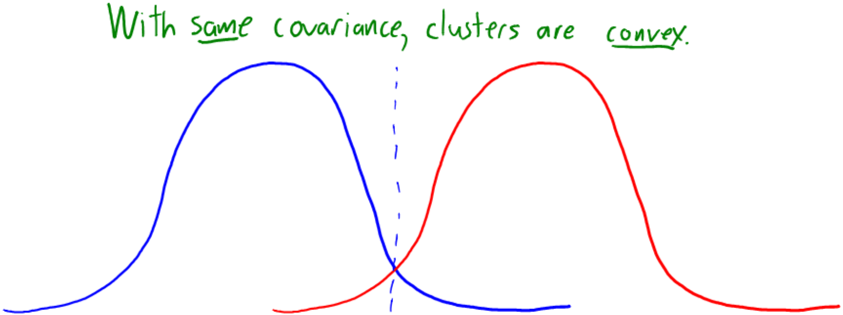
  - Step 1 is the learning step in GDA; Step 2 is the prediction step in GDA

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 Gaussian mixtures
  - May need to do multiple random restarts, or clever initializations like k-means++

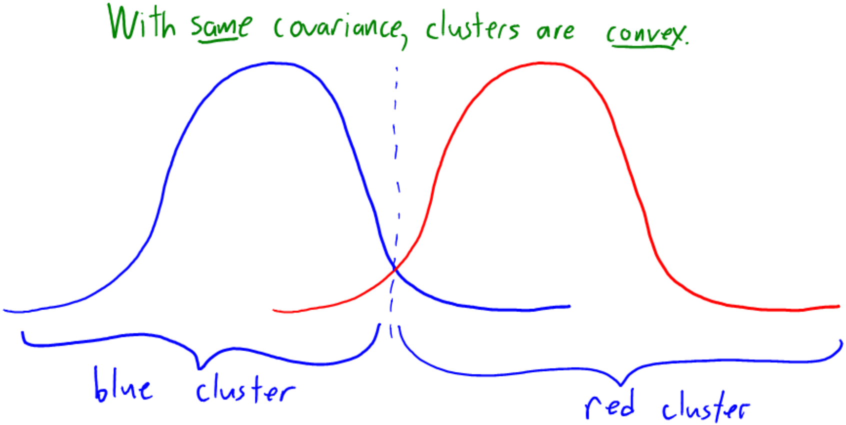
k-Means vs. Mixture of Gaussians

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



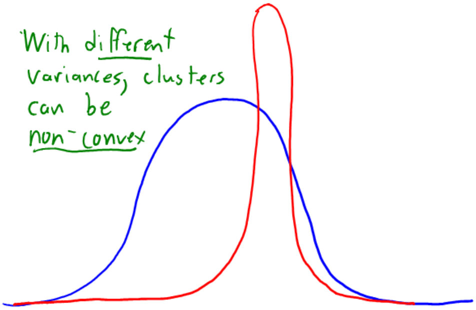
k-Means vs. Mixture of Gaussians

- 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



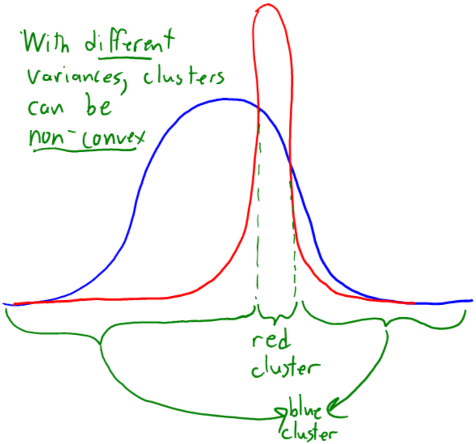
k-Means vs. Mixture of Gaussians

- 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



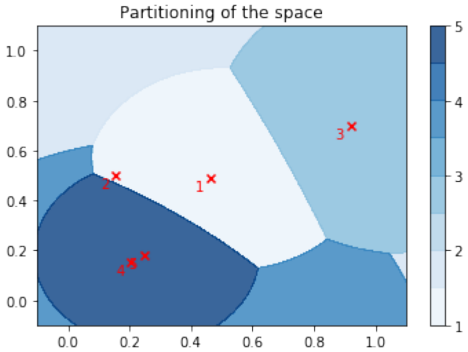
k-Means vs. Mixture of Gaussians

- 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



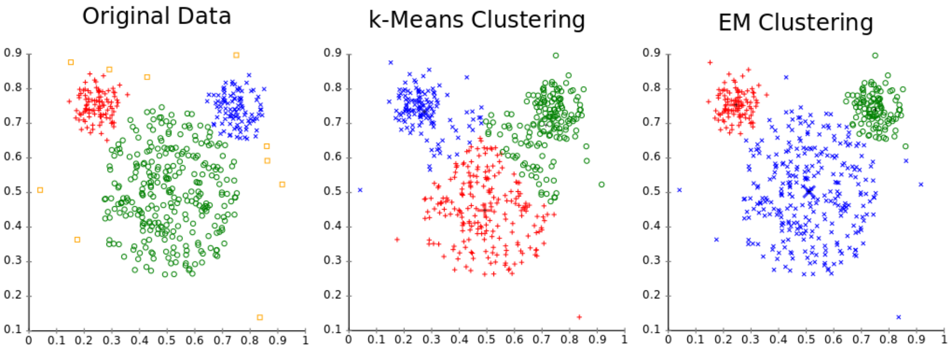
k-Means vs. Mixture of Gaussians

- 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 $\mathbf{\Sigma}_c \to 0$
  - Optimizers often find models with degenerate components
  - Also often get empty clusters
- It is common to remove empty clusters and use a regularized update,

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

which is MAP estimation with an L1 regularizer on diagonals of the precision

  - The MAP estimate exists with this and other usual priors on $\mathbf{\Sigma}_c$

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

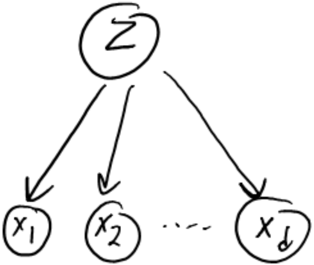
- Previous example: $\theta_{\cdot|1} = \begin{bmatrix} 0 & 0.7 & 1 & 1 \end{bmatrix}$ and $\theta_{\cdot|2} = \begin{bmatrix} 1 & 0.7 & 0.8 & 0 \end{bmatrix}$
- Here are some samples from this model:

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

- 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 $\theta_{j|c}$

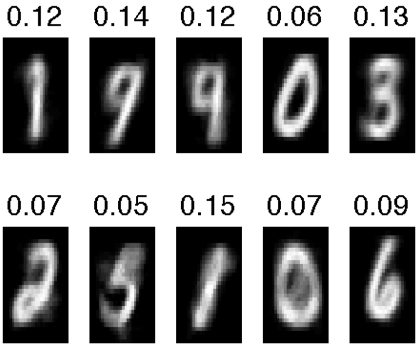
- Mixture of Bernoullis can model dependencies between variables
  - Individual mixtures act like clusters of the binary data
  - Knowing cluster of one variable gives information about other variables

- 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 $k \ll 2^d$...

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)

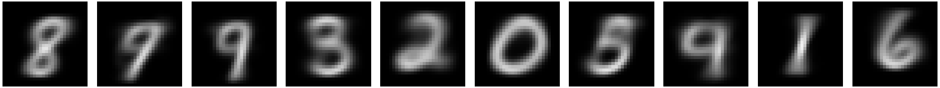


http://pmtk3.googlecode.com/svn/trunk/docs/demoOutput/bookDemos/%2811%29-Mixture_models_and_the_EM_algorithm/mixBerMnistEM.html

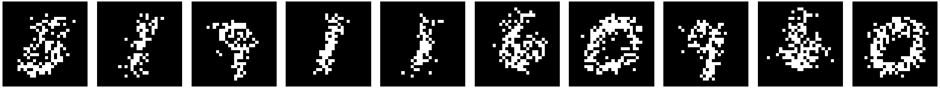
- Remember this is unsupervised: it hasn't been told there are ten digit classes
  - You could use this model to "fill in" missing parts of an image

Mixture of Bernoullis on Digits with $k > 10$

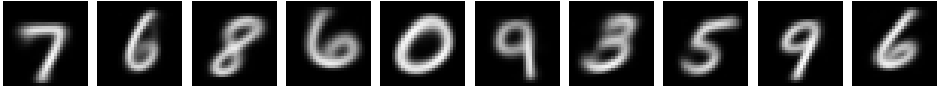
- Parameters of a mixture of Bernoulli model fit to MNIST with $k = 10$:



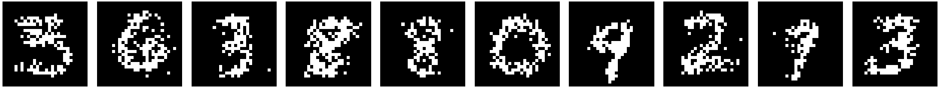
- Samples better than product of Bernoullis (but no within-cluster dependency):



- 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 Bernoullis:
  1. Find the responsibility of cluster $z^{(i)}$ for each example $x^{(i)}$:
  $$r_c^{(i)} = p(z^{(i)} = c \mid x^{(i)}, \Theta) \propto \pi_c p(x^{(i)} \mid z^{(i)} = c, \Theta)$$
  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 fitting mixture of Gaussians:
  1. Find the responsibility of cluster $z^{(i)}$ for each example $x^{(i)}$:

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

  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 how many examples the 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`

Fitting a Mixture of Exponential Families: Expectation Maximization

- Expectation maximization (EM) approach to fitting mixture of

$$p(x^{(i)} \mid z^{(i)} = c) = h(x^{(i)}) \exp\left(\theta_c^\mathsf{T} s\left(x^{(i)}\right)\right) / Z(\theta_c)$$

1. Find the responsibility of cluster $z^{(i)}$ for each example $x^{(i)}$:

$$r_c^{(i)} = p(z^{(i)} = c \mid x^{(i)}, \Theta) \propto \pi_c p(x^{(i)} \mid z^{(i)} = c, \Theta) \propto \pi_c \exp\left(\theta_c^\mathsf{T} s\left(x^{(i)}\right)\right) / Z(\theta_c)$$

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 parameters based on how many examples the cluster is responsible for:

$$\text{solve} \quad \mathop{\mathbb{E}}_{X \sim p_{\theta_c}} s(X) = \frac{1}{\sum_{i=1}^{n} r_c^{(i)}} \sum_{i=1}^{n} r_c^{(i)} s\left(x^{(i)}\right)$$

Expectation Maximization vs. Imputation

- The imputation method is optimizing $p(\mathbf{X}, \mathbf{Z} \mid \Theta)$ in terms of $\mathbf{Z}$ and Θ
  - $p(\mathbf{X}, \mathbf{Z} \mid \Theta)$ is called the complete-data likelihood
  - Steps are $\mathbf{Z}^{(t+1)} \in \arg\max_{\mathbf{Z}} p(\mathbf{X}, \mathbf{Z} \mid \Theta^{(t)})$, $\Theta^{(t+1)} \in \arg\max_{\Theta} p(\mathbf{X}, \mathbf{Z}^{(t+1)} \mid \Theta)$
  - Each step can only increase $p(\mathbf{X}, \mathbf{Z} \mid \Theta)$; finds a local max

- Expectation maximization (EM) is optimizing $p(\mathbf{X} \mid \Theta)$ in terms of Θ
  - So we're integrating over $\mathbf{Z}$ values while optimizing Θ
  - $p(\mathbf{X} \mid \Theta)$ is the usual likelihood, marginalizing over the $\mathbf{Z}$
  - But doing $\max_\Theta \mathbb{E}_{\mathbf{Z} \mid \mathbf{X}, \Theta} p(\mathbf{X}, \mathbf{Z} \mid \Theta)$ doesn't give us nice optimization tricks

$$\log \mathop{\mathbb{E}}_{\mathbf{Z} \mid \mathbf{X}, \Theta} p(\mathbf{X}, \mathbf{Z} \mid \Theta) = \log \mathop{\mathbb{E}}_{\mathbf{Z} \mid \mathbf{X}, \Theta} \prod_{i=1}^{n} \pi_{z^{(i)}} p(x^{(i)} \mid z^{(i)}, \Theta) = \sum_{i=1}^{n} \log \left(\mathop{\mathbb{E}}_{z^{(i)} \mid x^{(i)}, \Theta} \pi_{z^{(i)}} p(x^{(i)} \mid z^{(i)}, \theta) \right)$$

  - EM approximately maximizes this, as we'll see shortly

- 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 Algorithm: Properties

bonus!

- 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: More Properties

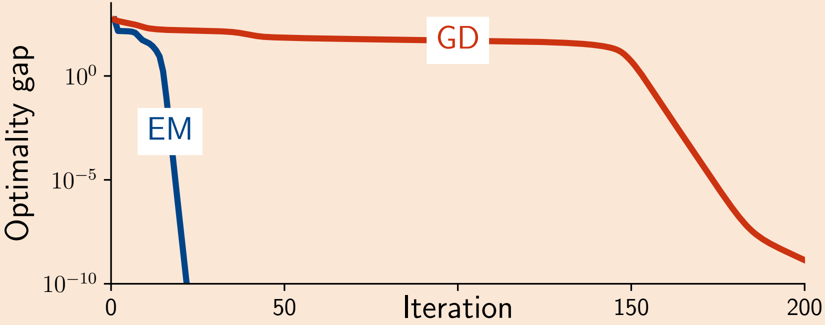
bonus!

- 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

bonus!

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



Outline

Missing data models

- In general, EM lets us do MLE/MAP with observed data $\mathbf{X}$ and missing data $\mathbf{Z}$
- Maybe we just didn't observe $x_j^{(i)}$...EM still lets us use the rest of x_j and $x^{(i)}$
- For mixture models, $\mathbf{Z}$ are the component IDs
- Related: class labels in semi-supervised learning, for "pseudo-labels"

The ELBO

- The Evidence Lower BOund is key to variational inference as well as EM

$$
\begin{aligned}
\log p(\mathbf{X} \mid \Theta) &= \int q(\mathbf{Z}) \log p(\mathbf{X} \mid \Theta) \mathrm{d}\mathbf{Z} \\
&= \int q(\mathbf{Z}) \log \left(\frac{p(\mathbf{X} \mid \Theta)\, p(\mathbf{Z} \mid \mathbf{X}, \Theta)}{p(\mathbf{Z} \mid \mathbf{X}, \Theta)} \right) \mathrm{d}\mathbf{Z} \\
&= \int q(\mathbf{Z}) \log \left(\frac{p(\mathbf{X}, \mathbf{Z} \mid \Theta)}{p(\mathbf{Z} \mid \mathbf{X}, \Theta)} \right) \mathrm{d}\mathbf{Z} = \int q(\mathbf{Z}) \log \left(\frac{p(\mathbf{X}, \mathbf{Z} \mid \Theta)\, q(\mathbf{Z})}{p(\mathbf{Z} \mid \mathbf{X}, \Theta)\, q(\mathbf{Z})} \right) \mathrm{d}\mathbf{Z} \\
&= \int q(\mathbf{Z}) \log p(\mathbf{X}, \mathbf{Z} \mid \Theta) \mathrm{d}\mathbf{Z} - \int q(\mathbf{Z}) \log q(\mathbf{Z}) \mathrm{d}\mathbf{Z} + \int q(\mathbf{Z}) \log \left(\frac{q(\mathbf{Z})}{p(\mathbf{Z} \mid \mathbf{X}, \Theta)} \right) \mathrm{d}\mathbf{Z} \\
&= \underbrace{\mathop{\mathbb{E}}_{\mathbf{Z} \sim q} \left[\log p(\mathbf{X}, \mathbf{Z} \mid \Theta) \right] + \mathrm{Entropy}[q]}_{\text{the ELBO}} + \mathrm{KL}(q(\mathbf{Z}) \parallel p(\mathbf{Z} \mid \mathbf{X}, \Theta))
\end{aligned}
$$

- $\mathrm{KL}(q \parallel p) \geq 0$ is the Kullback-Leibler divergence: zero iff $p = q$
- Tells us that $\mathrm{ELBO} \leq \log p(\mathbf{X} \mid \Theta)$ for any choice of distribution q

Information theory

- Entropy of a discrete random variable: $-\sum_x p(x) \log p(x) = \mathbb{E}_{X \sim p}[-\log p(X)]$
  - How efficiently can I encode a sample from p on average?
  - Entropy of a point mass is 0; of $\mathrm{Unif}(\{1, \dots, k\})$ is $-\log \frac{1}{k} = \log k$
- Differential entropy of a continuous rv: $-\int_x p(x) \log p(x) = \mathbb{E}_{X \sim p}[-\log p(X)]$
  - Can be negative! If $X \sim \mathrm{Unif}([0, 0.1])$, $\mathbb{E}[-\log p(X)] = -\log 10$
- KL divergence or relative entropy is $\mathrm{KL}(p \,\|\, q) = \mathbb{E}_{X \sim p}\left[\log \frac{p(X)}{q(X)}\right]$
  - How much do I lose by encoding a sample from p using a model for q?
  - $\mathrm{KL}(p \,\|\, q) = 0$ if $p = q$, otherwise positive: $f(x) = -\log(x)$ is convex, so (Jensen's)

$$\mathbb{E}\, f\left(\frac{q(x)}{p(x)}\right) \geq f\left(\mathbb{E}\, \frac{q(x)}{p(x)}\right) = -\log\left(\int_x \frac{q(x)}{p(x)} p(x) \mathrm{d}x\right) = -\log\left(\int_x q(x) \mathrm{d}x\right) = 0$$

  - Not symmetric: $\mathrm{KL}(p \,\|\, q) \neq \mathrm{KL}(q \,\|\, p)$ in general
- Cross-entropy: $\mathbb{E}_{X \sim p}[-\log q(X)] = \mathrm{Entropy}(p) + \mathrm{KL}(p \,\|\, q)$
  - How efficiently does a code for q encode a sample for p?

Applying ELBO grease

- We'd like to do $\max_\Theta \log p(\mathbf{X} \mid \Theta)$, but it's hard. For any distribution $q(z)$,

$$\log p(\mathbf{X} \mid \Theta) \geq \mathop{\mathbb{E}}_{\mathbf{Z} \sim q}[\log p(\mathbf{X}, \mathbf{Z} \mid \Theta)] + \text{Entropy}[q]$$

- If we choose Θ and q to get a large ELBO, we'd guarantee a large $\log p(\mathbf{X} \mid \Theta)$

$$\max_{\Theta, q} \mathop{\mathbb{E}}_{\mathbf{Z} \sim q} \log p(\mathbf{X}, \mathbf{Z} \mid \Theta) + \text{Entropy}[q]$$

- The bound is tight when $q(\mathbf{Z}) = p(\mathbf{Z} \mid \mathbf{X}, \Theta)$, since the KL term is zero:

$$\log p(\mathbf{X} \mid \Theta) = \mathop{\mathbb{E}}_{\mathbf{Z} \sim p(\mathbf{Z} \mid \mathbf{X}, \Theta)}[\log p(\mathbf{X}, \mathbf{Z} \mid \Theta)] + \text{Entropy}[p(\mathbf{Z} \mid \mathbf{X}, \Theta)]$$

  - So, for any Θ, the q that maximizes the ELBO is $p(\mathbf{Z} \mid \mathbf{X}, \Theta)$
- For any q, Θ maximizing ELBO is $\arg\max_\Theta \mathbb{E}_{\mathbf{Z} \sim q}[\log p(\mathbf{X}, \mathbf{Z} \mid \Theta)] + \text{Entropy}[q]$
- Alternate $q^{(t+1)} \in \arg\max_q \text{ELBO}(\Theta^{(t)}, q)$, $\Theta^{(t+1)} \in \arg\max_\Theta \text{ELBO}(\Theta, q^{(t+1)})$
  - Ends at local max of ELBO, which implies local max of $p(\mathbf{X} \mid \Theta)$
- Succinct statement of general EM: $\Theta^{(t+1)} \in \arg\max_\Theta \mathbb{E}_{\mathbf{Z} \mid \mathbf{X}, \Theta^{(t)}} \log p(\mathbf{X}, \mathbf{Z} \mid \Theta)$

EM for Mixture Models

- If $Z^{(i)} \overset{iid}{\sim} \text{Cat}(\pi)$ and $X^{(i)} \mid (Z^{(i)} = c) \sim \text{Something}(\theta_c)$,

$$\mathop{\mathbb{E}}_{\mathbf{Z}|\mathbf{X},\Theta^{(t)}} \log p(\mathbf{X}, \mathbf{Z} \mid \Theta) = \sum_{i=1}^{n} \mathop{\mathbb{E}}_{z^{(i)}|x^{(i)},\Theta^{(t)}} \log p(x^{(i)}, z^{(i)} \mid \Theta)$$
$$= \sum_{i=1}^{n} \sum_{c=1}^{k} p(z^{(i)} = c \mid x^{(i)}, \Theta^{(t)}) \left(\log \pi_c + \log p(x^{(i)} \mid z^{(i)} = c, \theta_c)\right)$$

- So, each EM iteration of finding Θ can be written as two steps:
  1. Expectation step: compute responsibilities $r_c^{(i)}$ for all i and c, for current $\Theta^{(t)}$
  2. Maximization step: maximize $\sum_i \sum_c r_c^{(i)} \log p(x^{(i)}, z^{(i)} \mid \Theta)$ by
     - Maximize over π_c: pick $\pi_c \propto \sum_i r_c^{(i)}$
     - Maximize over θ_c for each component, with "data weights" $r_c^{(i)}$

- Might not always implement with explicitly separate "E" and "M" steps
- EM best if $\mathbf{Z} \mid \mathbf{X}, \Theta$ is simple to compute, and $\log p(\mathbf{X}, \mathbf{Z} \mid \Theta)$ is easy to optimize

Outline

Combining Mixture Models with Other Models

bonus!

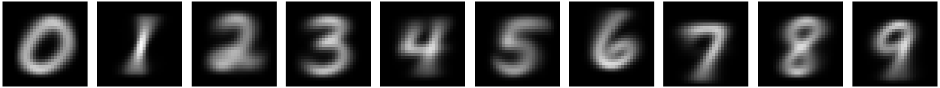
- We can use mixtures in generative models:
  - Model $p(x \mid y)$ as a mixture instead of simple Gaussian or product of Bernoullis
- Or in discriminative models:
  - Let $Y \mid X$ follow a mixture of Gaussians, with means chosen by a deep net

- 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, graphical models (later in the course)

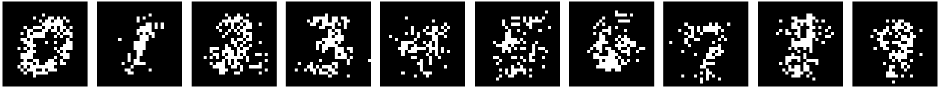
- We can add features to mixture models for supervised learning:
  - Mixture of experts: have k regression/classification models
    - Each model can be viewed as a "expert" for a cluster of $x^{(i)}$ values
    - GPT-4, Grok, . . . are mixtures of Transformers
    - These models use conditional weights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

Less-Naive Bayes on Digits

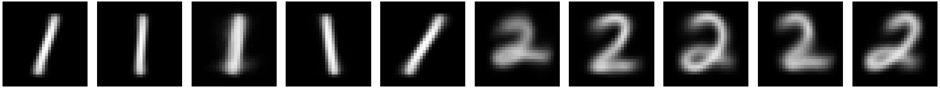
- Naive Bayes θ_c values (independent Bernoullis for each class):



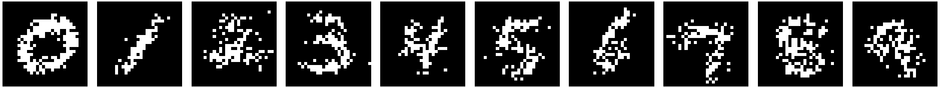
- One sample from each class:



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



- One sample from each class:



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

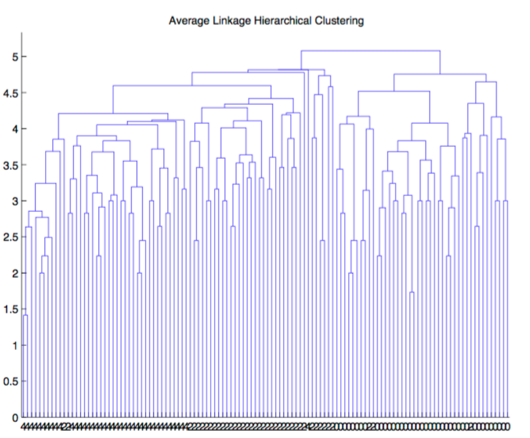
bonus!

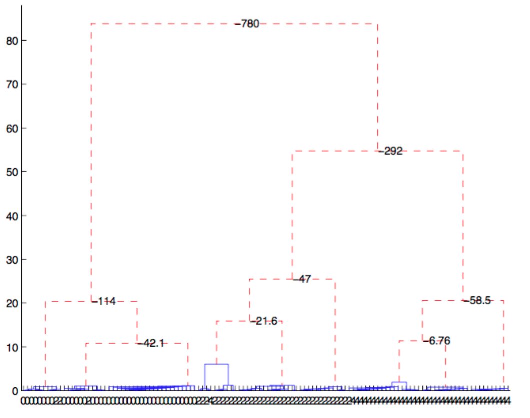
- Slides giving more details on Dirichelt process mixture models:
  - https://www.cs.ubc.ca/labs/lci/mlrg/slides/NP.pdf
- We could alternately put a prior on number of clusters k:
  - Allows more flexibility than Dirichlet process as a prior
  - Computationally more difficult
- There are a variety of 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:



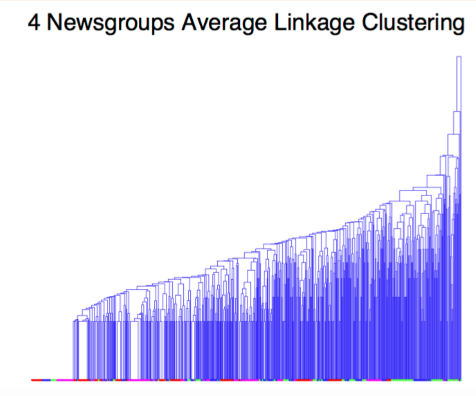


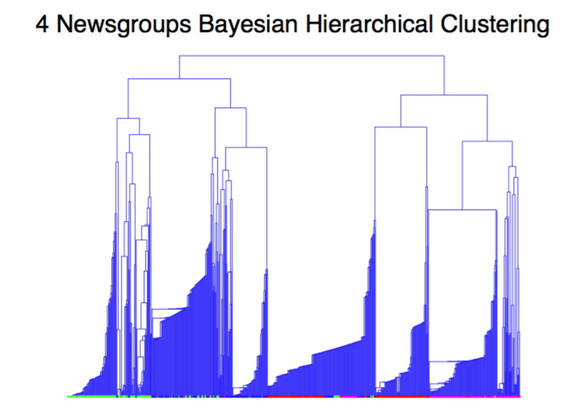
http://www2.stat.duke.edu/~kheller/bhcnew.pdf (y-axis represents distance between clusters)

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:

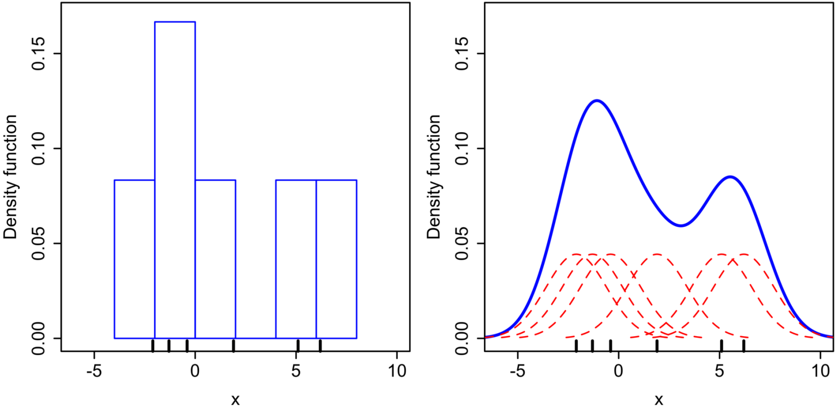
$$p(x^{(i)}) = \frac{1}{n}\sum_{j=1}^{n}\mathcal{N}(x^{(i)} \mid x^{(j)}, \sigma^2\mathbf{I})$$

- 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

- Number of components, means, mixture weights are fixed from $\mathbf{X}$; fitting is trivial
- 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

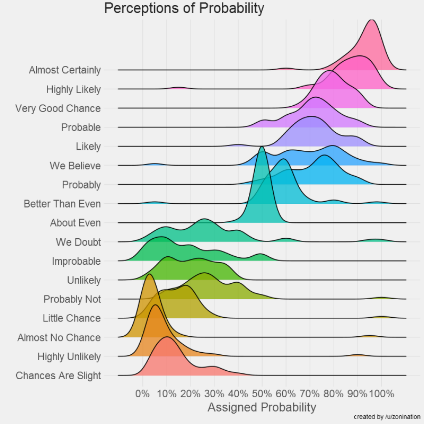
- You can think of a kernel density estimate as like a continuous histogram:



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.

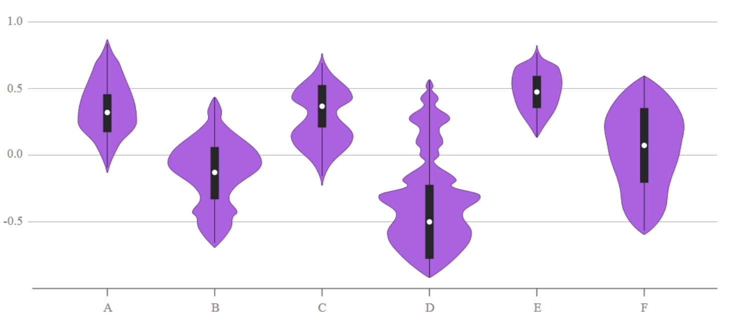


http://blog.revolutionanalytics.com/2017/08/probably-more-probably-than-probable.html

Violin Plot: Adding KDE to a Boxplot

bonus!

- Violin plot adds KDE to a boxplot:

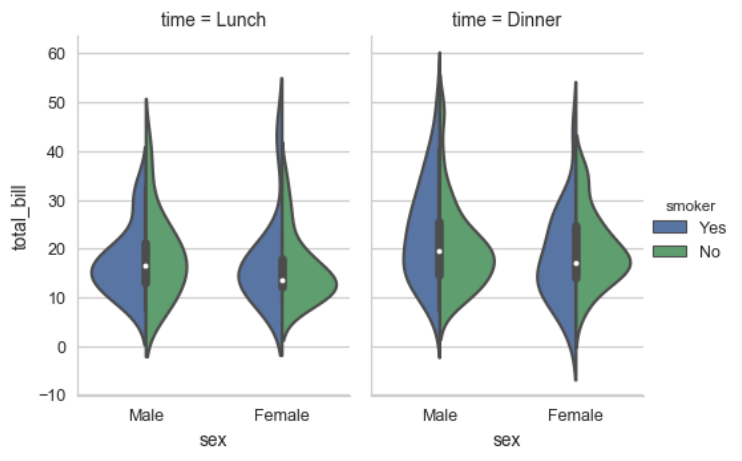


https://datavizcatalogue.com/methods/violin_plot.html

Violin Plot: Adding KDE to a Boxplot

bonus!

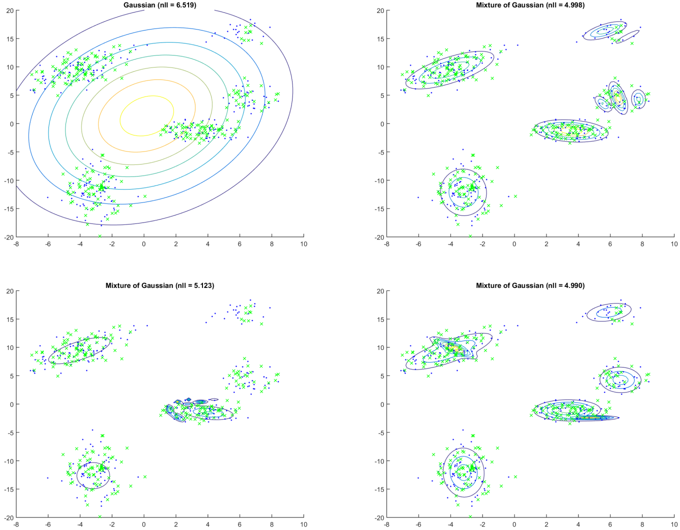
- Violin plot adds KDE to a boxplot:



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

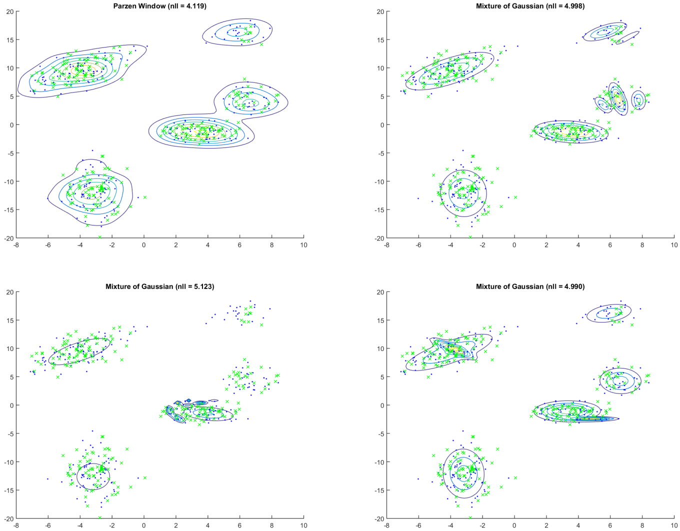
KDE vs. Mixture of Gaussian

- Single Gaussian 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

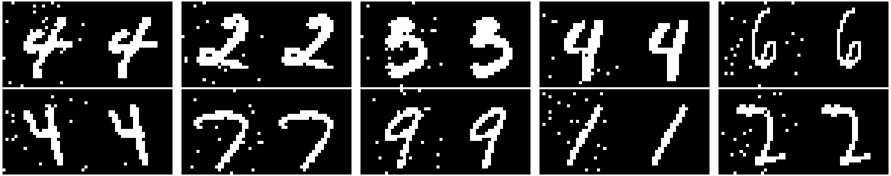
- Not sensitive to initialization, no need to choose number of clusters
- Can find non-convex clusters

- Similar to density-based clustering from 340
  - Doesn't require uniform density within cluster
  - Can be used for vector quantization

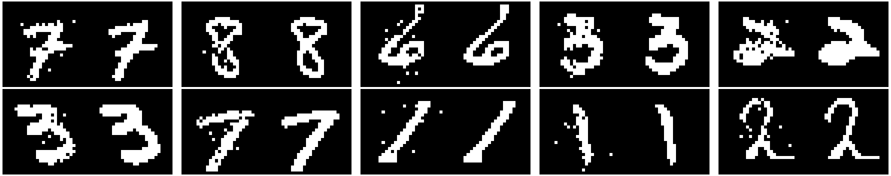
- "The 5 Clustering Algorithms Data Scientists Need to Know":
  - https://towardsdatascience.com/the-5-clustering-algorithms-data-scientists-need-to-know-a36d136ef68

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: hitting the casino

Avoiding Underflow when Computing Responsibilities

bonus!

- Computing responsibility may underflow for high-dimensional $x^{(i)}$, due to $p(x^{(i)} \mid z^{(i)} = c, \Theta)$

- Usual ML solution: do all but last step in log-domain

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

- 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:

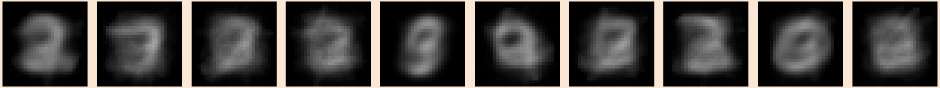
$$\begin{aligned}
\log\left(\sum_i \exp(v_i)\right) &= \log\left(\sum_i \exp(v_i - \beta + \beta)\right) \\
&= \log\left(\sum_i \exp(v_i - \beta)\exp(\beta)\right) \\
&= \log\left(\exp(\beta)\sum_i \exp(v_i - \beta)\right) \\
&= \log(\exp(\beta)) + \log\left(\sum_i \exp(v_i - \beta)\right) \\
&= \beta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

- Avoids overflows in computing the $\exp$ operator

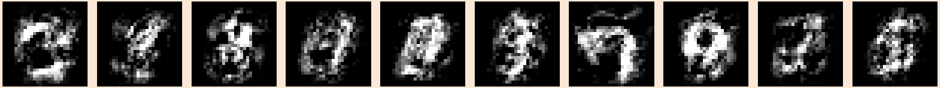
Mixture of Gaussians on Digits

bonus!

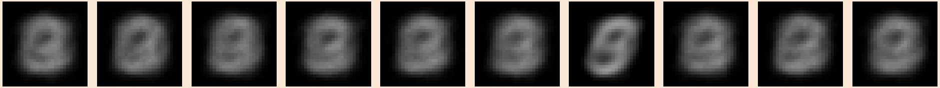
- Mean parameters of a mixture of Gaussians with $k = 10$:



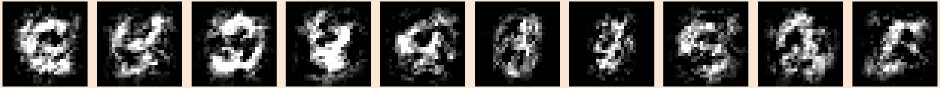
- Samples:



- 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,

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

- 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 discriminative 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.

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,

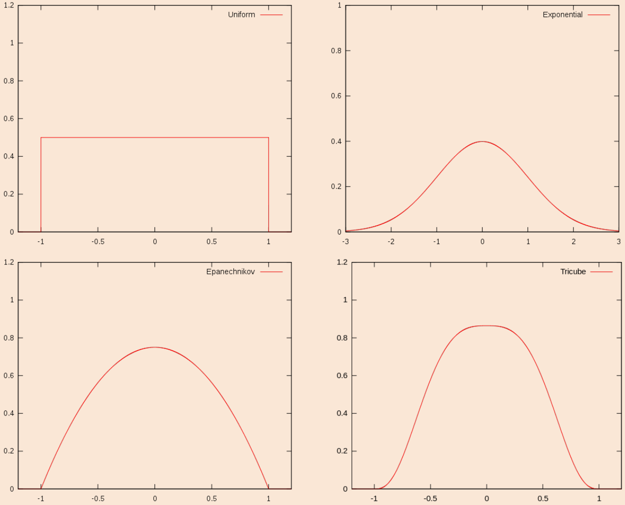
$$k_1(r) = \frac{3}{4}\left(1 - r^2\right)\mathcal{I}\left[|r| \leq 1\right].$$

- 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

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

- 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 $\Sigma = AA^T$
  - Can help, but choices other than $A = \sigma I$ add a lot of parameters!