master ATiAM

Acoustique
Traitement du signal
Informatique
Appliqués à la
Musique

A survey of Machine Learning techniques

Arshia Cont
Musical Representations Team, IRCAM.
cont@ircam.fr

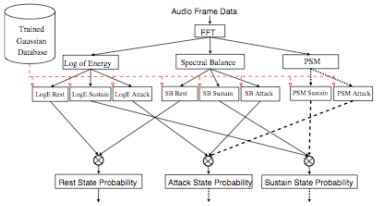
ATIAM 2011-12

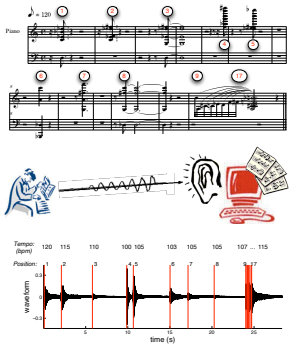
Plan

- Last time we saw:
  - Bayesian Decision Theory
  - Maximum Likelihood Parameter Estimations
  - Bayesian Parameter Estimation
- Today, we will look at:
  - Kernel Based Parameter Estimation
  - Mixture Models and EM Algorithm
  - Some Non-parametric methods
  - Sequential Learning
    - HMMs
    - Kalman Filters

Some musical examples

- The score following problem:
  - Need *observation models* at the front-end.
  - From audio frames to low-level state probs:

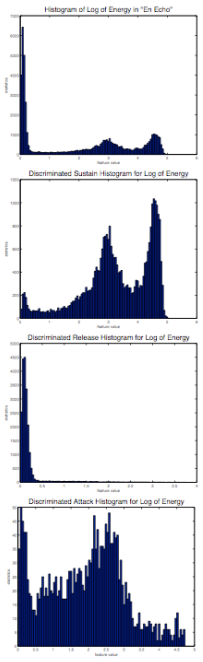




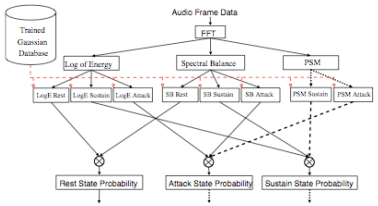
  - This was the model for *suivi* object (now defunct!)
- Problem 1: is how to train the generative probability models which give informative probabilities on Rest/Attack/Sustain

Example

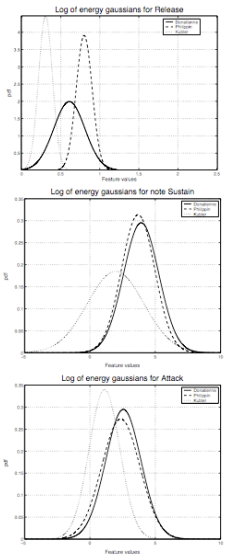
- Problem 2: These models will be probably different from performance to performance / musician to musician!
- Design cycle: Learning from rehearsal recordings
  - Gather segmented data
  - Design the generative models for each attribute
  - Train models from labeled database
  - Test!
  - Incorporate them in the realtime system for the performance.



Learned Gaussians for different performers>>>



<<<Observation Histograms



Kernel-based and non-parametric methods

master ATIAM '11

Plan

- We have already looked at Bayesian Decision Rules, and how to optimize them through Maximum-Likelihood (ML) or Bayesian Parameter Estimation... .
  - In all these formulations, we assume that X is generated by a probability density P(X)
- Practical densities do not approximate well using simple probability density families!
- We now look at ways to approach P(X) when the data is non-trivial or more complicated than a known and simple probability family... .
  - So far, we have considered *parametric* density estimations...
  - Today, we consider *non-parametric* density estimates...

Non-parametric density estimates

- Given iid training set $\mathcal{D} = \{\mathbf{x}_1, \ldots \mathbf{x}_n\}$, the goal is to estimate

$$P_{\mathbf{X}}(\mathbf{x})$$

- Consider a region $\mathcal{R}$, and define

$$P = P_{\mathbf{X}}[\mathbf{x} \in \mathcal{R}] = \int_{\mathcal{R}} P_{\mathbf{X}}(\mathbf{x}) d\mathbf{x}.$$

and define

$$K = \sharp\{\mathbf{x}_i \in \mathcal{D} | \mathbf{x}_i \in \mathcal{R}\}.$$

- This is a binomial distribution of paramter P

$$\begin{aligned} P_K(k) &= \mathcal{B}(n, P) \\ &= \binom{n}{k} P^k (1-P)^{n-k} \end{aligned}$$

Binomial random variable

- ML estimate of P

$$\hat{P} = \frac{k}{n}.$$

and statistiscs

$$\begin{aligned} E[\hat{P}] &= \frac{1}{n}E[k] = \frac{1}{n}nP = P \\ var[\hat{P}] &= \frac{1}{n^2}var[k] = \frac{P(1-P)}{n}. \end{aligned}$$

- Note that $var[\hat{P}] \leq 1/4n$ goes to zero very quickly, i.e.

$$\hat{P} \to P.$$

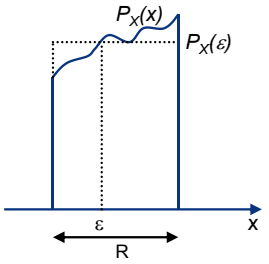
N	10	100	1,000	…
Var[P] <	0.025	0.0025	0.00025	

Histogram

- this means that *k/n* is a very good estimate of *P*
- on the other hand, from the mean value theorem, if $P_X(x)$ is continuous $\exists \epsilon \in \mathcal{R}$ such that

$$P = \int_{\mathcal{R}} P_{\mathbf{X}}(\mathbf{x})d\mathbf{x} = P_{\mathbf{X}}(\epsilon) \int_{\mathcal{R}} d\mathbf{x} = P_{\mathbf{X}}(\epsilon)V(\mathcal{R}).$$

- this is easiest to see in 1D
  - can always find a box such that the integral of the function is equal to that of the box
  - since $P_X(x)$ is continuous there must be a ε such that $P_X(\varepsilon)$ is the box height



Histogram

- hence

$$P_{\mathbf{X}}(\epsilon) = \frac{P}{V(\mathcal{R})} \approx \frac{\hat{P}}{V(\mathcal{R})} = \frac{k}{nV(\mathcal{R})}$$

- using continuity of $P_X(x)$ again and assuming *R* is small

$$P_{\mathbf{X}}(\mathbf{x}) \approx \frac{k}{nV(\mathcal{R})}, \ \forall \mathbf{x} \in V(\mathcal{R})$$

- this is the histogram
- it is the simplest possible non-parametric estimator
- can be generalized into kernel-based density estimator

Kernel density estimates

- assume $\mathcal{R}$ is the d-dimensional cube of side h

$$V = h^d$$

and define *indicator* function of the unit hypercube

$$\phi(\mathbf{u}) = \begin{cases} 1, & \text{if } |u_i| < 1/2 \\ 0, & \text{otherwise.} \end{cases}$$

hence

$$\phi\left(\frac{\mathbf{x} - \mathbf{x}_i}{h}\right) = 1$$

iif $\mathbf{x}_i \in$ hypercube of volume V centered at $\mathbf{x}$.

- the number of sample points in the hypercube is

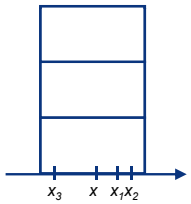
$$k_n = \sum_{i=1}^{n} \phi\left(\frac{\mathbf{x} - \mathbf{x}_i}{h}\right)$$

Kernel density estimates

- this means that the histogram can be written as

$$P_{\mathbf{X}}(\mathbf{x}) = \frac{1}{nh^d}\sum_{i=1}^{n}\phi\left(\frac{\mathbf{x}-\mathbf{x}_i}{h}\right)$$

- which is equivalent to:
  - "put a box around X for each X_i that lands on the hypercube"
  - can be seen as a very crude form of interpolation
  - better interpolation if contribution of X_i decreases with distance to X
- consider other windows $\phi(x)$



Windows

- what sort of functions are valid windows?
- note that $P_X(x)$ is a pdf if and only if

$$P_{\mathbf{X}}(\mathbf{x}) \geq 0, \forall \mathbf{x} \text{ and } \int P_{\mathbf{X}}(\mathbf{x})d\mathbf{x} = 1$$

- since

$$\begin{aligned}\int P_{\mathbf{X}}(\mathbf{x})d\mathbf{x} &= \frac{1}{nh^d}\sum_{i=1}^{n}\int\phi\left(\frac{\mathbf{x}-\mathbf{x}_i}{h}\right)d\mathbf{x}\\ &= \frac{1}{nh^d}\sum_{i=1}^{n}\int\phi(\mathbf{y})\,h^d d\mathbf{y}\\ &= \frac{1}{n}\sum_{i=1}^{n}\int\phi(\mathbf{y})\,d\mathbf{y}\end{aligned}$$

- these conditions hold if $\phi(x)$ is itself a pdf

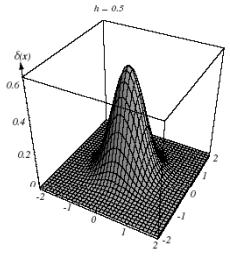
$$\phi(\mathbf{x}) \geq 0, \forall \mathbf{x} \text{ and } \int \phi(\mathbf{x})d\mathbf{x} = 1$$

Gaussian kernel

- probably the most popular in practice

$$\phi(\mathbf{x}) = \frac{1}{\sqrt{2\pi}^d}e^{-\frac{1}{2}\mathbf{x}^T\mathbf{x}}$$

- note that $P_X(x)$ can also be seen as a sum of pdfs centered on the X_i when $\phi(x)$ is symmetric in X and X_i

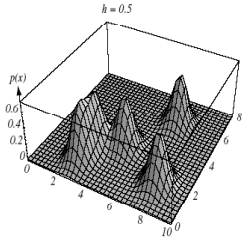


$$P_{\mathbf{X}}(\mathbf{x}) = \frac{1}{nh^d}\sum_{i=1}^{n}\phi\left(\frac{\mathbf{x}-\mathbf{x}_i}{h}\right)$$

Gaussian kernel

- Gaussian case can be interpreted as
  - sum of n Gaussians centered at the X_i with covariance $h\mathbf{I}$
  - more generally, we can have a full covariance

$$P_{\mathbf{X}}(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{\sqrt{(2\pi)^d|\Sigma|}}e^{-\frac{1}{2}(\mathbf{x}-\mathbf{x}_i)^T\Sigma^{-1}(\mathbf{x}-\mathbf{x}_i)}$$



- sum of n Gaussians centered at the X_i with covariance Σ
- Gaussian kernel density estimate: *"approximate the pdf of X with a sum of Gaussian bumps"*

Kernel bandwidth

- back to the generic model

$$P_{\mathbf{X}}(\mathbf{x}) = \frac{1}{nh^d}\sum_{i=1}^{n}\phi\left(\frac{\mathbf{x}-\mathbf{x}_i}{h}\right)$$

- what is the role of *h* (bandwidth parameter)?
- defining

$$\delta(\mathbf{x}) = \frac{1}{h^d}\phi\left(\frac{\mathbf{x}}{h}\right)$$

- we can write

$$P_{\mathbf{X}}(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n}\delta\left(\mathbf{x}-\mathbf{x}_i\right)$$

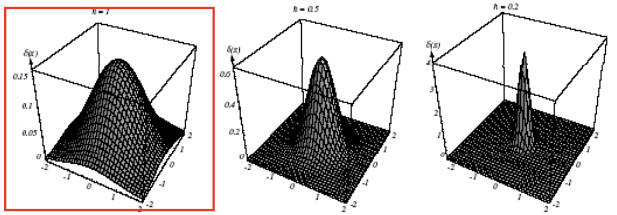
- i.e. a sum of translated replicas of $\delta(x)$

Kernel bandwidth

- *h* has two roles:
  1. rescale the *x*-axis
  2. rescale the amplitude of $\delta(x)$

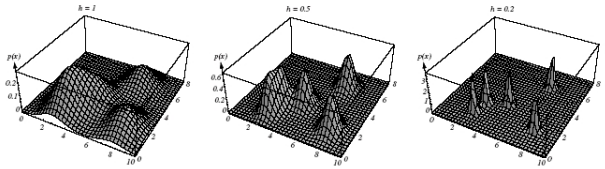
$$\delta(\mathbf{x}) = \frac{1}{h^d}\phi\left(\frac{\mathbf{x}}{h}\right)$$

- this implies that for large *h*:
  1. $\delta(x)$ has low amplitude
  2. iso-contours of *h* are quite distant from zero (*x* large before $\phi(x/h)$ changes significantly from $\phi(0)$)



Kernel bandwidth

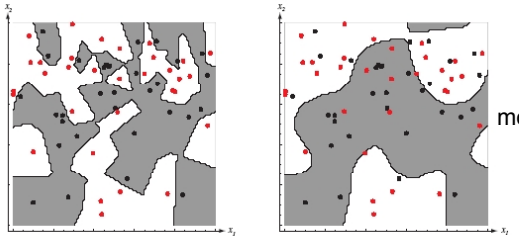
- it controls the smoothness of the estimate
  - as h goes to zero we have a sum of delta functions (very "spiky" approximation)
  - as h goes to infinity we have a sum of constant functions (approximation by a constant)
  - in between we get approximations that are gradually more smooth



Kernel bandwidth

- why does this matter?
- when the density estimates are plugged into the BDR
- smoothness of estimates determines the smoothness of the boundaries

less smooth

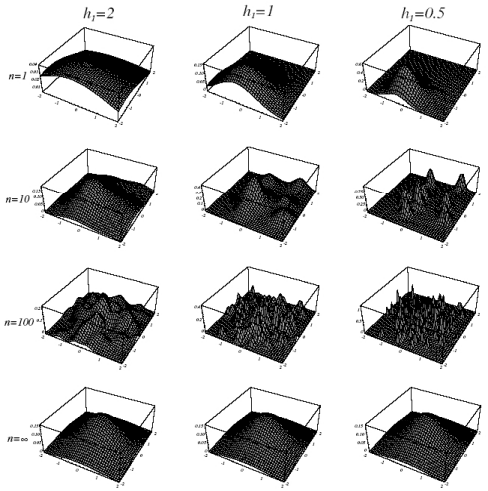


more smooth

- this affects the probability of error!

Example

- example: fit to N(0,I) using $h = h_1/n^{1/2}$
- small h: spiky
- need a lot of points to converge (variance)
- large h: approximate N(0,I) with a sum of Gaussians of larger covariance
- will never have zero error (bias)

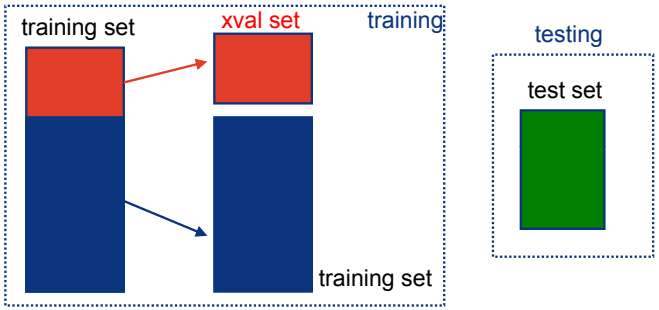


Optimal bandwidth

- in practice this has limitations
  - does not say anything about the finite data case (the one we care about)
  - still have to find the best k
- usually we end up using trial and error or techniques like cross-validation

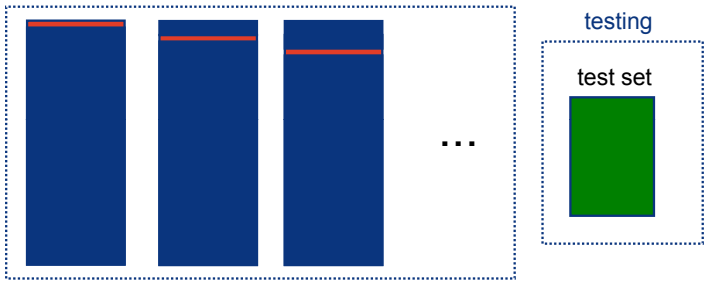
Cross-validation

- basic idea:
  - leave some data out of your training set (cross validation set)
  - train with different parameters
  - evaluate performance on cross validation set
  - pick best parameter configuration



Leave-one-out cross-validation

- many variations
- leave-one-out CV:
  - compute n estimators of $P_X(x)$ by leaving one X_i out at a time
  - for each $P_X(x)$ evaluate $P_X(X_i)$ on the point that was left out
  - pick $P_X(x)$ that maximizes this likelihood

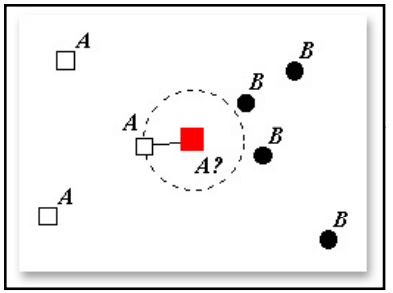


Non-parametric classifiers

- given kernel density estimates for all classes we can compute the BDR
- since the estimators are non-parametric the resulting classifier will also be non-parametric
- this term is general and applies to any learning algorithm
- a very simple example is the nearest neighbor classifier

Nearest neighbor classifier

- is the simplest possible classifier that one could think of:
  - it literally consists of assigning to the vector to classify the label of the closest vector in the training set
  - to classify the red point:
    - measure the distance to all other points
    - if the closest point is a square, assign to "square" class
    - otherwise assign to "circle" class

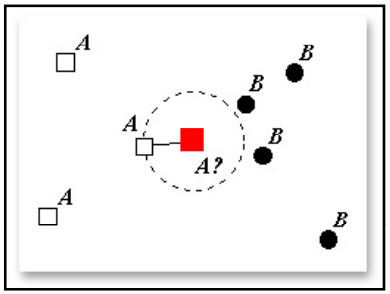


- it works a lot better than what one might predict

Nearest neighbor classifier

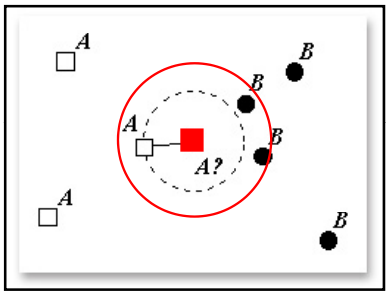
- to define it mathematically we need to define
  - a training set $\mathcal{D} = \{(x_1,y_1), \ldots, (x_n,y_n)\}$
  - x_i is a vector of observations, y_i is the label
  - a vector x to classify
- the "decision rule" is

$$set \quad y = y_{i^*}$$
$$where$$
$$i^* = \arg\min_{i \in \{1,\ldots,n\}} d(x, x_i)$$



k-nearest neighbors

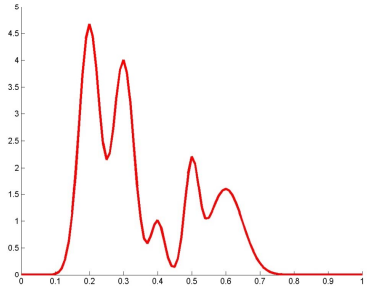
- instead of the NN, assigns to the majority vote of the k nearest neighbors
- in this example
  - NN rule says "A"
  - but 3-NN rule says "B"
- for x away from the border does not make much difference



- usually best performance for k > 1, but there is no universal number
- k large: performance degrades (no longer neighbors)
- k should be odd, to prevent ties

Mixture density estimates

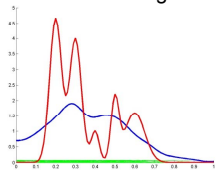
- back to BDR-based classifiers
  Consider the problem of instrument classification
- summary:
  - Estimate instrument type (brass, string, percu) from audio
  - Measure some audio feature
  - estimate pdf
  - use BDR
- clearly this is not Gaussian
- possible solution: use a kernel-based model



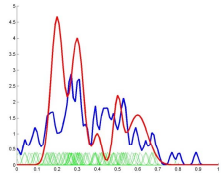
Kernel-based estimate

- simple learning procedure
  - measure audio feature X
  - place a Gaussian on top of each measurement
- can be overkill
  - spending all degrees of freedom (# of training points) just to get the Gaussian means
  - cannot use the data to determine variances
- handpicking of bandwidth can lead to too much bias or variance

bandwidth too large: bias



bandwidth too small: variance

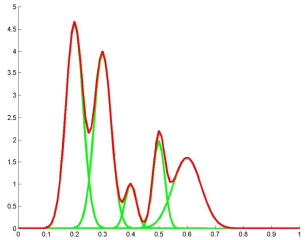


mixture density estimate

- it looks like we could do better by just picking the right # of Gaussians
- this is indeed a good model:
  - density is multimodal because there is a hidden variable Z
    - Z can determine the type of intermediate musical instruments (for example)

$$Z \in \{Violin, Piano, Saxophone, Flute, Drum\}$$

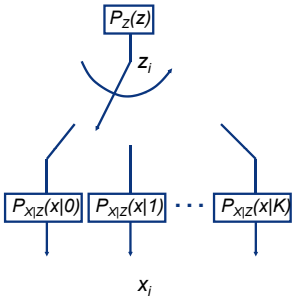
    - Note that this is different from *Y* which is the instrument type (brass, string, percussion)
    - For a given instrument type, the density is approximate Gaussian here.
    - The density is a *mixture of Gaussians*



mixture model

- two types of random variables
  - *Z* – hidden state variable
  - *X* – observed variable
- observations sampled with a two-step procedure
  - a state (class) is sampled from the distribution of the hidden variable

$$P_Z(z) \rightarrow z_i$$

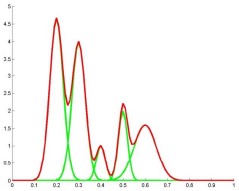


  - an observation is drawn from the class conditional density for the selected state

$$P_{X|Z}(x|z_i) \rightarrow x_i$$

mixture model

- the sample consists of pairs (x_i, z_i)
  $D = \{(x_1, z_1), \ldots, (x_n, z_n)\}$
  but we never get to see the z_i



- the pdf of the observed data is

$$P_{\mathbf{X}}(\mathbf{x}) = \sum_{c=1}^{C} P_{\mathbf{X}|Z}(\mathbf{x}|c) P_Z(c) = \sum_{c=1}^{C} P_{\mathbf{X}|Z}(\mathbf{x}|c)\pi_c$$

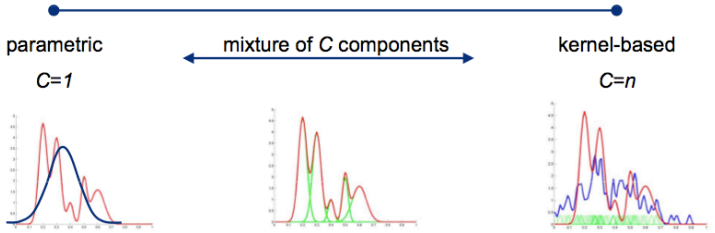
# of mixture components

component "weight"

c^{th} "mixture component"

Mixtures vs Kernel and parametric

- A parametric model is a mixture with *one* component
  - The weight is *one*
  - The mixture density is the parametric density itself!
  - More degrees of freedom in mixture => less bias
- A mixture density is like a kernel density less components
  - less components => less learning parameters, less variance
- Mixture is a compromise between these two extremes:



mixture disadvantages

- main disadvantage is learning complexity
- non-parametric estimates
  - simple: store the samples (NN); place a kernel on top of each point (kernel-based)
- parametric estimates
  - small amount of work: if ML equations have closed-form
  - substantial amount of work: otherwise (numerical solution)
- mixtures:
  - there is usually no closed-form solution
  - always need to resort to numerical procedures
- standard tool is the expectation-maximization (EM) algorithm

Clustering and EM algorithm

Classes vs. Clusters

- Supervised: $\mathcal{X} = \{ \mathbf{x}^t, \mathbf{r}^t \}_t$
- Classes C_i i=1,...,K

$$p(\mathbf{x}) = \sum_{i=1}^{K} p(\mathbf{x} \mid C_i) P(C_i)$$

where $p(\mathbf{x} \mid C_i) \sim \mathcal{N}(\mu_i, \Sigma_i)$

- $\Phi = \{P(C_i), \mu_i, \Sigma_i\}_{i=1}^{K}$

$$\hat{P}(C_i) = \frac{\sum_t r_i^t}{N} \quad \mathbf{m}_i = \frac{\sum_t r_i^t \mathbf{x}^t}{\sum_t r_i^t}$$

$$\mathbf{S}_i = \frac{\sum_t r_i^t (\mathbf{x}^t - \mathbf{m}_i)(\mathbf{x}^t - \mathbf{m}_i)^T}{\sum_t r_i^t}$$

- Unsupervised : $\mathcal{X} = \{ \mathbf{x}^t \}_t$
- Clusters $\mathcal{G}_i$ i=1,...,k

$$p(\mathbf{x}) = \sum_{i=1}^{k} p(\mathbf{x} \mid \mathcal{Z}_i) P(\mathcal{Z}_i)$$

where $p(\mathbf{x} \mid \mathcal{Z}_i) \sim \mathcal{N}(\mu_i, \Sigma_i)$

- $\Phi = \{P(\mathcal{Z}_i), \mu_i, \Sigma_i\}_{i=1}^{k}$

Labels, $\mathbf{r}^t_i$?

K-Means Clustering

- Dataset $D = \{x_1, x_2, \ldots, x_n\}$
- Goal: Partition in K clusters
  - Cluster prototype: μ_k
  - Binary indicator variable (1-of-K coding scheme)

$$r_{nk} \in \{0, 1\}$$
$$r_{nk} = 1, \quad \text{and } r_{nj} = 0 \text{for } j \neq k$$

hard assignment

  - Distortion measure

$$J = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} ||x_n - \mu_k||^2$$

- K-means gives k reference vectors (prototypes) which can be used as decision rule
  - Our sample communication problem:

$$E(\{\mathbf{m}_i\}_{i=1}^{k} | \mathcal{X}) = \sum_t \sum_i b_i^t \|\mathbf{x}^t - \mathbf{m}_i\|$$

$$b_i^t = \begin{cases} 1 & \text{if } \|\mathbf{x}^t - \mathbf{m}_i\| = \min_j \|\mathbf{x}^t - \mathbf{m}_j\| \\ 0 & \text{otherwise} \end{cases}$$

K-Means clustering: EM

- Find values for $\{r_{nk}\}$ and $\{\mu_k\}$ to minimize:

$$J = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} ||x_n - \mu_k||^2$$

- Iterative procedure:
  - Expectation: Minimize J with regards to $\{r_{nk}\}$, keep $\{\mu_k\}$ fixed

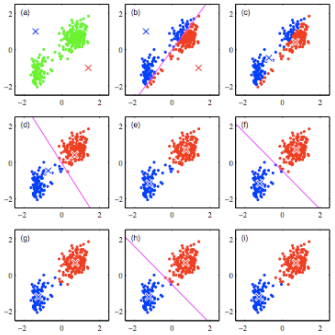
$$r_{nk} = \begin{cases} 1, & \text{if } k = \arg\min_j ||x_n - \mu_k||^2 \\ 0, & \text{otherwise} \end{cases}$$

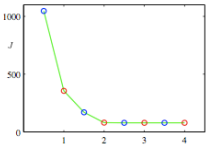
  - Maximization: Minimize J with regards to $\{\mu_k\}$, keep $\{r_{nk}\}$ fixed

$$2 \sum_{n=1}^{N} r_{nk} (x_n - \mu_k) = 0$$

$$\mu_k = \frac{\sum_n r_{nk} x_n}{\sum_n r_{nk}}$$

K-Means Clustering: Example





- Each E or M step reduces the value of the objective function J
- Convergence to a global or local maximum

K-means and clustering

- Problems:
  - How many clusters? (K)
    - Various methods available: *Bayesian Information Criterion, Akaike Information Criterion, Minimum Description Length*
    - Or guessing + cross-validation!
  - Local minimum only
    - Can be a source of head-ache!
  - Initialization of the means
    - Another source of head-ache!
    - Usual method: *mean-splitting*...
- *Sounds great..*
  - But what about non-classification problems?!

The basics of EM

- as usual, we start from an iid sample $D = \{x_1, \ldots, x_N\}$
- goal is to find parameters $\Psi^\star$ that maximize likelihood with respect to D

$$\Psi^\star = \arg\max_{\Psi} P_{\mathbf{X}}(\mathcal{D}; \Psi)$$
$$= \arg\max_{\Psi} \int P_{\mathbf{X}|Z}(\mathcal{D}|z; \Psi) P_Z(z; \Psi) dz$$

- the set

$$D_c = \{(x_1, z_1), \ldots, (x_N, z_N)\}$$

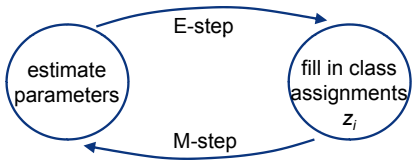
  is called the complete data
- the set

$$D = \{x_1, \ldots, x_N\}$$

  is called the incomplete data

Learning with incomplete data (EM)

- the basic idea is quite simple
  1. start with an initial parameter estimate $\Psi^{(0)}$
  2. **E-step:** given current parameters $\Psi^{(i)}$ and observations in D, "guess" what the values of the z_i are
  3. **M-step:** with the new z_i, we have a complete data problem, solve this problem for the parameters, i.e. compute $\Psi^{(i+1)}$
  4. go to 2.
- this can be summarized as



Classification-maximization

- C-step:
  - given estimates $\Psi^{(i)} = \{\Psi^{(i)}_1, \ldots, \Psi^{(i)}_C\}$
  - determine z_i by the BDR

$$z_l = \arg\max_c P_{\mathbf{X}|Z}\left(\mathbf{x}_l|c; \Psi_c^{(i)}\right) \pi_c^{(i)}, l \in \{1, \ldots, n\}$$

  - split the training set according to the labels z_i

    $D^1 = \{x_i|z_i=1\}$, $D^2 = \{x_i|z_i=2\}$, ... , $D^C = \{x_i|z_i=C\}$
- M-step:
  - as before, determine the parameters of each class independently

$$\Psi_c^{(i+1)} = \arg\max_{\Psi,\pi} P_{\mathbf{X}|Z}(\mathcal{D}^c|c, \Psi)\pi$$

For Gaussian mixtures

- C-step:
  - $z_l = \arg\max_c \left\{ -\frac{1}{2}\left(\mathbf{x}_l - \mu_c^{(i)}\right)^T \left(\Sigma_c^{(i)}\right)^{-1} \left(\mathbf{x}_l - \mu_c^{(i)}\right) - \frac{1}{2} log \left|\Sigma_c^{(i)}\right| + \log \pi_c^{(i)} \right\}, l \in \{1, \ldots, n\}$
  - split the training set according to the labels z_i

    $D^1 = \{x_i | z_i = 1\}, \quad D^2 = \{x_i | z_i = 2\}, \quad \ldots \quad , D^C = \{x_i | z_i = C\}$

- M-step:
  - $\pi_c^{(i+1)} = \frac{|\{\mathbf{x}_i \in \mathcal{D}^c\}|}{n} \qquad \mu_c^{(i+1)} = \frac{1}{|\{\mathbf{x}_i \in \mathcal{D}^c\}|} \sum_{i|\mathbf{x}_i \in \mathcal{D}^c} \mathbf{x}_i$

  $\Sigma_c^{(i+1)} = \frac{1}{|\{\mathbf{x}_i \in \mathcal{D}^c\}|} \sum_{i|\mathbf{x}_i \in \mathcal{D}^c} \left(\mathbf{x}_i - \mu_c^{(i+1)}\right)\left(\mathbf{x}_i - \mu_c^{(i+1)}\right)^T$

K-means

- when covariances are identity and priors uniform
- C-step:
  - $z_l = \arg\min_c ||\mathbf{x}_l - \mu_c^{(i)}||^2, \qquad l \in \{1, \ldots, n\}$
  - split the training set according to the labels z_i

    $D^1 = \{x_i | z_i = 1\}, \quad D^2 = \{x_i | z_i = 2\}, \quad \ldots \quad , D^C = \{x_i | z_i = C\}$
- M-step:
  - $\mu_c^{(i+1)} = \frac{1}{|\{\mathbf{x}_i \in \mathcal{D}^c\}|} \sum_{i|\mathbf{x}_i \in \mathcal{D}^c} \mathbf{x}_i$
- this is the K-means algorithm, aka generalized Loyd algorithm, aka LBG algorithm in the vector quantization literature:
  - "assign points to the closest mean; recompute the means"

Expectation-Maximization

- What about problems that are not about classification?
- EM suggests:
  - Do the most intuitive operation that is ALWAYS possible
  - Don't worry about Z_i directly
  - E-Step: "estimate the likelihood of the complete data by its expected value given the observed data"
  - M-step: "Maximize this expected value"
  - This leads to the so called *Q-function*

The Q function

- is defined as

  $Q(\Psi; \Psi^{(n)}) = E_{Z|\mathbf{X};\Psi^{(n)}} \left[\log P_{\mathbf{X},Z}(\mathcal{D}, \{z_1, \ldots, z_N\}; \Psi) | \mathcal{D}\right]$
- and is a bit tricky:
  - it is the expected value of likelihood with respect to complete data (joint X and Z)
  - given that we observed incomplete data (X=$\mathcal{D}$)
  - note that the likelihood is a function of Ψ (the parameters that we want to determine)
  - but to compute the expected value we need to use the parameter values from the previous iteration (because we need a distribution for Z|X)
- the EM algorithm is, therefore, as follows

Expectation-maximization

- E-step:
  - given estimates $\Psi^{(n)} = \{\Psi^{(n)}_1, \ldots, \Psi^{(n)}_C\}$
  - compute expected log-likelihood of complete data

$$Q(\Psi; \Psi^{(n)}) = E_{Z|\mathbf{X};\Psi^{(n)}} \left[\log P_{\mathbf{X},Z}(\mathcal{D}, \{z_1, \ldots, z_N\}; \Psi) | \mathcal{D} \right]$$

- M-step:
  - find parameter set that maximizes this expected log-likelihood

$$\Psi^{(n+1)} = \arg\max_{\Psi} Q(\Psi; \Psi^{(n)})$$

- let's make this more concrete by looking at the mixture case

Expectation-maximization

- to derive an EM algorithm you need to do the following
1. write down the likelihood of the COMPLETE data
2. E-step: write down the Q function, i.e. its expectation given the observed data
3. M-step: solve the maximization, deriving a closed-form solution if there is one

EM for mixtures (step 1)

- the first thing we always do in a EM problem is
  - compute the likelihood of the COMPLETE data
- very neat trick to use when z is discrete (classes)
  - instead of using *z in {1, 2, ..., C}*
  - use a binary vector of size equal to the # of classes

$$\mathbf{z} \in \left\{ \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}, \ldots, \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix} \right\}$$

  - where *z = j* in the *z in {1, 2, ..., C}* notation, now becomes

$$\mathbf{z} = \mathbf{e}_j = \begin{bmatrix} 0 \\ \vdots \\ 1 \quad (j^{th} position) \\ \vdots \\ 0 \end{bmatrix}$$

EM for mixtures (step 1)

- we can now write the complete data likelihood as

$$\begin{aligned} P_{\mathbf{X},\mathbf{Z}}(\mathbf{x}, \mathbf{z}; \Psi) &= P_{\mathbf{X}|\mathbf{Z}}(\mathbf{x}|\mathbf{z}; \Psi) P_{\mathbf{Z}}(\mathbf{z}; \Psi) \\ &= \prod_{j=1}^{C} \left[P_{\mathbf{X}|\mathbf{Z}}(\mathbf{x}|\mathbf{e}_j, \Psi) \pi_j \right]^{z_j} \end{aligned}$$

- for example, if *z = k* in the *z in {1, 2, ..., C}* notation,

$$\begin{aligned} P_{\mathbf{X},Z}(\mathbf{x}, k; \Psi) &= P_{\mathbf{X},\mathbf{Z}}(\mathbf{x}, \mathbf{e}_k; \Psi) \\ &= \left[P_{\mathbf{X}|\mathbf{Z}}(\mathbf{x}|\mathbf{e}_k, \Psi) \pi_k \right]^1 \prod_{j \neq k} \left[P_{\mathbf{X}|\mathbf{Z}}(\mathbf{x}|\mathbf{e}_j, \Psi) \pi_j \right]^0 \end{aligned}$$

- the advantage is that

$$\log P_{\mathbf{X},\mathbf{Z}}(\mathbf{x}, \mathbf{z}; \Psi) = \sum_{j=1}^{C} z_j \log \left[P_{\mathbf{X}|\mathbf{Z}}(\mathbf{x}|\mathbf{e}_j, \Psi) \pi_j \right]$$

- becomes LINEAR in the components z_j !!!

EM for mixtures (step 1)

- for the complete iid dataset $D_c = \{(x_1,z_1), \ldots, (x_N,z_N)\}$

$$P_{\mathbf{X},Z}(\mathcal{D}, \{\mathbf{z}_1, \ldots, \mathbf{z}_N\}; \Psi) = \prod_{i=1}^{N} P_{\mathbf{X},Z}(\mathbf{x}_i, \mathbf{z}_i; \Psi)$$
$$= \prod_{i=1}^{N} \prod_{j=1}^{C} \left[P_{\mathbf{X}|\mathbf{Z}}(\mathbf{x}_i|\mathbf{e}_j, \Psi)\pi_j\right]^{z_{ij}}$$

- and the complete data log-likelihood is

$$\log P_{\mathbf{X},Z}(\mathcal{D}, \{\mathbf{z}_1, \ldots, \mathbf{z}_N\}; \Psi) = \sum_{i,j} z_{ij} \log \left[P_{\mathbf{X}|\mathbf{Z}}(\mathbf{x}_i|\mathbf{e}_j, \Psi)\pi_j\right]$$

- this does not depend on z and simply becomes a constant for the expectation that we have to compute in the E-step

Expectation-maximization

- to derive an EM algorithm you need to do the following
  1. write down the likelihood of the COMPLETE data
  2. E-step: write down the Q function, i.e. its expectation given the observed data
  3. M-step: solve the maximization, deriving a closed-form solution if there is one
- important E-step advice:
  - do not compute terms that you do not need
  - at the end of the day we only care about the parameters
  - terms of Q that do not depend on the parameters are useless, e.g. in

    Q = f(z, Ψ) + log(sin z)

    the expected value of *log(sin z)* appears to be difficult and is completely unnecessary, since it is dropped in the M-step

EM for mixtures (step 2)

- once we have the complete data likelihood

$$Q(\Psi; \Psi^{(n)}) = E_{Z|\mathbf{X};\Psi^{(n)}}\left[\log P_{\mathbf{X},Z}(\mathcal{D}, \{z_1, \ldots, z_N\}; \Psi)|\mathcal{D}\right]$$
$$= \sum_{i,j} E_{Z|\mathbf{X};\Psi^{(n)}}[z_{ij}|\mathcal{D}] \log \left[P_{\mathbf{X}|\mathbf{Z}}(\mathbf{x}_i|\mathbf{e}_j, \Psi)\pi_j\right]$$

- i.e. to compute the Q function we only need to compute

$$E_{Z|\mathbf{X};\Psi^{(n)}}[z_{ij}|\mathcal{D}], \ \forall i,j$$

- note that this expectation can only be computed because we use $\Psi^{(n)}$
- note that the Q function will be a function of both Ψ and $\Psi^{(n)}$

EM for mixtures (step 2)

- since z_{ij} is binary and only depends on x_i

$$E_{\mathbf{Z}|\mathbf{X};\Psi^{(n)}}[z_{ij}|\mathcal{D}] = P_{\mathbf{Z}|\mathbf{X}}(z_{ij} = 1|\mathbf{x}_i; \Psi^{(n)}) = P_{\mathbf{Z}|\mathbf{X}}(\mathbf{e}_j|\mathbf{x}_i; \Psi^{(n)})$$

- the E-step reduces to computing the posterior probability of each point under each class!
- defining

$$h_{ij} = P_{\mathbf{Z}|\mathbf{X}}(\mathbf{e}_j|\mathbf{x}_i; \Psi^{(n)})$$

- the Q function is

$$Q(\Psi; \Psi^{(n)}) = \sum_{i,j} h_{ij} \log \left[P_{\mathbf{X}|\mathbf{Z}}(\mathbf{x}_i|\mathbf{e}_j, \Psi)\pi_j\right]$$

Expectation-maximization

- to derive an EM algorithm you need to do the following
  1. write down the likelihood of the COMPLETE data

$$\log P_{\mathbf{X},Z}(\mathcal{D}, \{\mathbf{z}_1, \ldots, \mathbf{z}_N\}; \Psi) = \sum_{i,j} z_{ij} \log \left[P_{\mathbf{X}|\mathbf{Z}}(\mathbf{x}_i|\mathbf{e}_j, \Psi)\pi_j \right]$$

  2. E-step: write down the Q function, i.e. its expectation given the observed data

$$h_{ij} = P_{\mathbf{Z}|\mathbf{X}}(\mathbf{e}_j|\mathbf{x}_i; \Psi^{(n)})$$

$$Q(\Psi; \Psi^{(n)}) = \sum_{i,j} h_{ij} \log \left[P_{\mathbf{X}|\mathbf{Z}}(\mathbf{x}_i|\mathbf{e}_j, \Psi)\pi_j \right]$$

  3. M-step: solve the maximization, deriving a closed-form solution if there is one

$$\Psi^{(n+1)} = \arg\max_{\Psi} \sum_{ij} h_{ij} \log \left[P_{\mathbf{X}|\mathbf{Z}}(\mathbf{x}_i|\mathbf{e}_j, \Psi)\pi_j \right]$$

EM for Gaussian mixtures

- in summary:
  - CM = EM + hard assignments
  - CM special case, cannot be better
- let's look at the special case of Gaussian mixtures
- **E-step:**

$$\begin{aligned} h_{ij} &= P_{\mathbf{Z}|\mathbf{X}}(\mathbf{e}_j|\mathbf{x}_i; \Psi^{(n)}) \\ &= \frac{\mathcal{G}\left(\mathbf{x}_i, \mu_j^{(n)}, \sigma_j^{(n)}\right)\pi_j^{(n)}}{\sum_{k=1}^{C} \mathcal{G}\left(\mathbf{x}_i, \mu_k^{(n)}, \sigma_k^{(n)}\right)\pi_k^{(n)}} \end{aligned}$$

M-step for Gaussian mixtures

- **M-step:**

$$\begin{aligned} \Psi^{(n+1)} &= \arg\max_{\Psi} \sum_{ij} h_{ij} \log \left[\mathcal{G}\left(\mathbf{x}_i, \mu_j, \sigma_j\right)\pi_j \right] \\ &= \arg\min_{\Psi} \sum_{ij} \frac{h_{ij}(\mathbf{x}_i - \mu_j)^2}{2\sigma_j^2} + \frac{h_{ij}}{2} \log \sigma_j^2 - h_{ij} \log \pi_j \end{aligned}$$

- important note:
  - in the M-step, the optimization must be subject to whatever constraint may hold
  - in particular, we always have the constraint $\sum_j \pi_j = 1$
  - as usual we introduce a Lagrangian

$$L = \sum_{ij} \left[\frac{h_{ij}(\mathbf{x}_i - \mu_j)^2}{2\sigma_j^2} + \frac{h_{ij}}{2} \log \sigma_j^2 - h_{ij} \log \pi_j \right] + \lambda \left(\sum_j \pi_j - 1 \right)$$

Not familiar with Lagrange multipliers? See http://en.wikipedia.org/wiki/Lagrange_multipliers

M-step for Gaussian mixtures

- Lagrangian

$$L = \sum_{ij} \left[\frac{h_{ij}(\mathbf{x}_i - \mu_j)^2}{2\sigma_j^2} + \frac{h_{ij}}{2} \log \sigma_j^2 - h_{ij} \log \pi_j \right] + \lambda \left(\sum_j \pi_j - 1 \right)$$

- setting derivatives to zero

$$\begin{aligned} \frac{\partial L}{\partial \mu_j} &= -\sum_i \frac{h_{ij}(\mathbf{x}_i - \mu_j)}{\sigma_j^2} = 0 \\ \frac{\partial L}{\partial \sigma_j^2} &= -\sum_i \left[\frac{h_{ij}(\mathbf{x}_i - \mu_j)^2}{\sigma_j^4} - \frac{h_{ij}}{\sigma_j^2} \right] = 0 \\ \frac{\partial L}{\partial \pi_j} &= -\sum_i \frac{h_i}{\pi_j} + \lambda = 0 \\ \frac{\partial L}{\partial \lambda} &= \sum_j \pi_j - 1 = 0 \end{aligned}$$

M-step for Gaussian mixtures

- leads to the update equations

$$\mu_j^{(n+1)} = \frac{\sum_i h_{ij}\mathbf{x}_i}{\sum_i h_{ij}} \qquad \pi_j^{(n+1)} = \frac{1}{n}\sum_i h_{ij}$$

$$\sigma_j^{2(n+1)} = \frac{\sum_i h_{ij}(\mathbf{x}_i - \mu_j)^2}{\sum_i h_{ij}}$$

- comparing to those of CM

$$\pi_c^{(n+1)} = \frac{|\{\mathbf{x}_i \in \mathcal{D}^c\}|}{N} \qquad \mu_c^{(n+1)} = \frac{1}{|\{\mathbf{x}_i \in \mathcal{D}^c\}|}\sum_{i|\mathbf{x}_i \in \mathcal{D}^c} \mathbf{x}_i$$

$$\Sigma_c^{(n+1)} = \frac{1}{|\{\mathbf{x}_i \in \mathcal{D}^c\}|}\sum_{i|\mathbf{x}_i \in \mathcal{D}^c} \left(\mathbf{x}_i - \mu_c^{(n+1)}\right)\left(\mathbf{x}_i - \mu_c^{(n+1)}\right)^T$$

- they are the same up to hard vs soft assignments.

Expectation-maximization

- note that the procedure is the same for all mixtures
  1. write down the likelihood of the COMPLETE data

$$\log P_{\mathbf{X},Z}(\mathcal{D}, \{\mathbf{z}_1, \ldots, \mathbf{z}_N\}; \Psi) = \sum_{i,j} z_{ij} \log \left[P_{\mathbf{X}|\mathbf{Z}}(\mathbf{x}_i|\mathbf{e}_j, \Psi)\pi_j\right]$$

  2. E-step: write down the Q function, i.e. its expectation given the observed data

$$h_{ij} = P_{\mathbf{Z}|\mathbf{X}}(\mathbf{e}_j|\mathbf{x}_i; \Psi^{(n)})$$

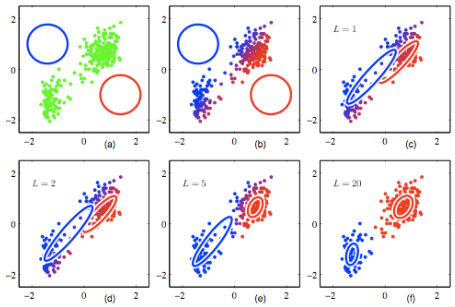
$$Q(\Psi; \Psi^{(n)}) = \sum_{i,j} h_{ij} \log \left[P_{\mathbf{X}|\mathbf{Z}}(\mathbf{x}_i|\mathbf{e}_j, \Psi)\pi_j\right]$$

  3. M-step: solve the maximization, deriving a closed-form solution if there is one

$$\Psi^{(n+1)} = \arg\max_{\Psi} \sum_{ij} h_{ij} \log \left[P_{\mathbf{X}|\mathbf{Z}}(\mathbf{x}_i|\mathbf{e}_j, \Psi)\pi_j\right]$$

EM on Gaussian Mixtures

- Example:



Group Homework 3

- Derive the EM algorithm for a mixture of exponential distributions:

$$P_X(x) = \sum_{i=1}^{C} \pi_i \lambda_i e^{-\lambda_i x}$$

  - Write down the E-step
  - Write down the M-step, solve the maximization and derive iterative solutions for λ_k and π_k
    - hint: Use the Lagrangian.... .

Sequential Learning

master ATiAM '11

Sequential Learning

- Up to now, our approach was rather *static*. Now imagine that you have a problem set where data arrives *sequentially*. *(e.g. Time Series)*
  - *A sequence of observations:* $y_1, y_2, y_3, \ldots, y_t$
  - Considered as independent... :

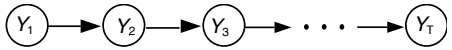
$$P(y_1, y_2, \ldots, y_t) = \prod_{n=1}^{t} p(y_n | y_1, \ldots, y_{n-1})$$

- Many many ways to model sequential data! We will look at some... .

Markov Models

First-order Markov model:

$$P(\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_t) = P(\mathbf{y}_1) P(\mathbf{y}_2 | \mathbf{y}_1) \cdots P(\mathbf{y}_t | \mathbf{y}_{t-1})$$



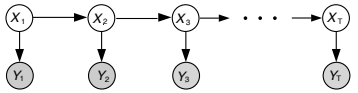
The term *Markov* refers to a conditional independence relationship. In this case, the Markov property is that, given the present observation ($\mathbf{y}_t$), the future ($\mathbf{y}_{t+1}, \ldots$) is independent of the past ($\mathbf{y}_1, \ldots, \mathbf{y}_{t-1}$).

Second-order Markov model:

$$P(\mathbf{y}_1, \ldots, \mathbf{y}_t) = P(\mathbf{y}_1) P(\mathbf{y}_2 | \mathbf{y}_1) \cdots P(\mathbf{y}_t | \mathbf{y}_{t-2}, \mathbf{y}_{t-1})$$

Hidden Variables

Speech recognition:
- $\mathbf{x}$ - underlying phonemes or words
- $\mathbf{y}$ - acoustic waveform

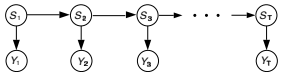


Vision:
- $\mathbf{x}$ - object identities, poses, illumination
- $\mathbf{y}$ - image pixel values

Industrial Monitoring:
- $\mathbf{x}$ - current state of molten steel in caster
- $\mathbf{y}$ - temperature and pressure sensor readings

Two frequently-used tractable models:
- Linear-Gaussian state-space models
- Hidden Markov models

Hidden Markov Models (HMM)



- Discrete hidden states $s_t \in \{1 \dots, K\}$, and outputs $\mathbf{y}_t$ (discrete or continuous). Joint probability factorizes:

$$\mathrm{P}(s_1, \dots, s_\tau, \mathbf{y}_1 \dots, \mathbf{y}_\tau) = \mathrm{P}(s_1)\mathrm{P}(\mathbf{y}_1|s_1) \prod_{t=2}^{\tau} \mathrm{P}(s_t|s_{t-1})\mathrm{P}(\mathbf{y}_t|s_t)$$

- Leading to the following important structural elements:
  - Transition probabilities:

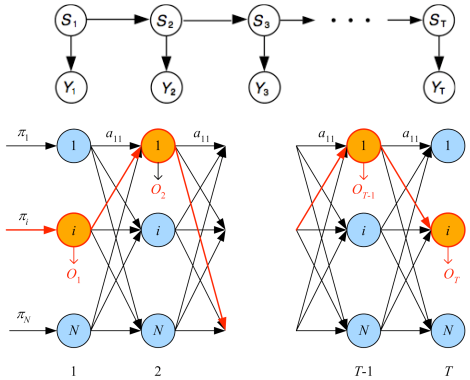
$$a_{ij} = P(s_t = S_j | s_{t-1} = S_i) \,, \quad a_{ij} \geq 0 \; and \; \sum_{j=1}^{N} a_{ij} = 1$$

  - Observation probabilities
    - Discrete case: $b_j(m) = P(y_t = Y_m | s_t = S_j)$
  - Initial probabilities (prior):

$$\pi_i = P(s_1 = S_i) \,, \quad \sum_{j=1}^{N} \pi_j = 1$$

Hidden Markov Models (HMM)

- Notes:
  - Hidden states (S_i) are markov... but not necessarily the output process (Y_i). In reality, they are NOT markov... .
  - For each observation sequence, there are multiple state sequences



- N: Number of states
- M: Number of observation symbols
- $\mathbf{A} = [a_{ij}]$: N by N state transition proba. matrix
- $\mathbf{B} = b_j(m)$: N by M observation proba. matrix
- $\Pi = [\pi_i]$: N by 1 initial state proba. vector

$\lambda = (\mathbf{A}, \mathbf{B}, \Pi)$, parameter set of HMM

HMMs

Three basic HMM problems:

1. Evaluation: Given λ, and O, calculate $P(O \mid \lambda)$
2. State sequence: Given λ, and O, find Q^* such that

$$P(Q^* \mid O, \lambda) = \max_Q P(Q \mid O, \lambda)$$

3. Learning: Given $\mathcal{X}=\{O^k\}_k$, find λ^* such that

$$P(\mathcal{X} \mid \lambda^*) = \max_\lambda P(\mathcal{X} \mid \lambda)$$

Likelihood in HMM

- The likelihood $P(O_1, \dots, O_\tau | \lambda)$ is an extremely hard computation
  - Number of possible paths grow exponentially with time (# of paths=K^τ)
- To compute this likelihood, there exist an efficient *forward* recursion algorithm using dynamic programming:

$$\alpha_t(i) = P(O_1, \dots, O_t, s_t = S_i | \lambda)$$

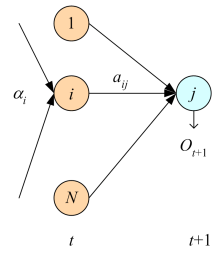
Initialization :

$$\alpha_1(i) = \pi_i b_i(O_1)$$

Recursion :

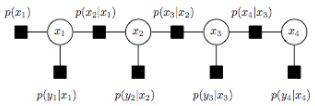
$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i) a_{ij} \right] b_j(O_{t+1})$$

$$P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i)$$



Likelihood in HMM

or:



$$p(y_{1:K}) = \sum_{x_{1:K}} p(y_{1:K}|x_{1:K})p(x_{1:K})$$

$$= \sum_{x_K} p(y_K|x_K) \sum_{x_{K-1}} p(x_K|x_{K-1}) \cdots \sum_{x_2} p(x_3|x_2)p(y_2|x_2) \sum_{x_1} p(x_2|x_1) \underbrace{p(y_1|x_1)}_{\alpha_{1|1}} \overbrace{p(x_1)}^{\alpha_{1|0}}$$

$$= \sum_{x_K} p(y_K|x_K) \sum_{x_{K-1}} p(x_K|x_{K-1}) \cdots \sum_{x_2} p(x_3|x_2)p(y_2|x_2) \sum_{x_1} p(x_2|x_1)\alpha_{1|1}(x_1)$$

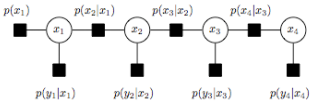
$$= \sum_{x_K} p(y_K|x_K) \sum_{x_{K-1}} p(x_K|x_{K-1}) \cdots \sum_{x_2} p(x_3|x_2)p(y_2|x_2)\alpha_{2|1}(x_2)$$

$$= \sum_{x_K} p(y_K|x_K) \sum_{x_{K-1}} p(x_K|x_{K-1}) \cdots \sum_{x_2} p(x_3|x_2)\alpha_{2|2}(x_2)$$

$$= \sum_{x_K} p(y_K|x_K) \sum_{x_{K-1}} p(x_K|x_{K-1}) \cdots \alpha_{3|2}(x_3)$$

Likelihood in HMM

- Note:



- Predict

$$\alpha_{k|k-1}(x_k) = p(y_{1:k-1}, x_k) = \sum_{x_{k-1}} p(x_k|x_{k-1})p(y_{1:k-1}, x_{k-1})$$

$$= \sum_{x_{k-1}} p(x_k|x_{k-1})\alpha_{k-1|k-1}(x_{k-1})$$

- Update

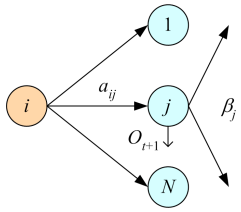
$$\alpha_{k|k}(x_k) = p(y_{1:k}, x_k) = p(y_k|x_k)p(y_{1:k-1}, x_k)$$

$$= p(y_k|x_k)\alpha_{k|k-1}(x_k)$$

Most likely state-sequence

- Forward variable gives the likelihood of an observation sequence given model parameters or $P(O|\lambda)$
- What about the most-likely sequence or $P(S^*|O,\lambda) = \max_S P(S|O,\lambda)$
  - *Forward* variable is the most-likely belief from time *0* to *t* (t=present)
  - Need also a belief towards future (from time t to T)
  - *Backward* variable:

$$\beta_t(i) = P(O_{t+1}, \ldots, O_T, s_t = S_i|\lambda)$$



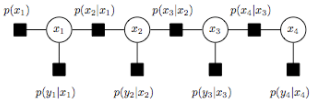
Initialization :

$$\beta_T(i) = 1$$

Recursion :

$$\beta_t(j) = \left[\sum_{i=1}^{N} \alpha_{t+1}(i)a_i j\right] b_j(O_{t+1})$$

- Another view:



$$p(y_{1:K}) = \sum_{x_1} p(x_1)p(y_1|x_1) \cdots \sum_{x_{K-1}} p(x_{K-1}|x_{K-2})p(y_{K-1}|x_{K-1}) \sum_{x_K} p(x_K|x_{K-1})p(y_K|x_K) \underbrace{1}_{\beta_{K|K+1}}$$

$$= \sum_{x_1} p(x_1)p(y_1|x_1) \cdots \sum_{x_{K-1}} p(x_{K-1}|x_{K-2})p(y_{K-1}|x_{K-1}) \sum_{x_K} p(x_K|x_{K-1})\beta_{K|K}$$

$$= \sum_{x_1} p(x_1)p(y_1|x_1) \cdots \sum_{x_{K-1}} p(x_{K-1}|x_{K-2})p(y_{K-1}|x_{K-1})\beta_{K-1|K}$$

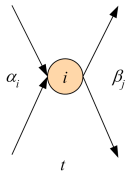
$$= \sum_{x_1} p(x_1)p(y_1|x_1) \cdots \sum_{x_{K-1}} p(x_{K-1}|x_{K-2})\beta_{K-1|K-1}$$

$$= \sum_{x_1} p(x_1)p(y_1|x_1) \ldots \beta_{K-2|K-1}$$

Most-likely state-sequence

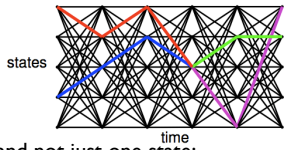
- Combine the two propagations and use maximum likelihood?

$$\begin{aligned} s_t(i) &= P(s_t = S_i|O,\lambda) \\ &= \frac{\alpha_t(i)\beta_t(i)}{\sum_{j=1}^{N}\alpha_t(j)\beta_t(j)} \\ s_t^* &= \arg\max_i s_t(i) \end{aligned}$$



Problem:

- Is the set of locally optimal states, equal to the global optimal path?
  - NO! not necessarily... .
    - $\alpha_i(t)$ gives total *inflow* of prob. to node (t,i); $\beta_i(t)$ gives total *outflow* of prob.



- Need to maximize over the WHOLE PATH and not just one state:

$$s_t(i) = \max_{s_1 s_2 \cdots s_{t-1}} P(s_1, s_2 \cdots s_{t-1}, s_t = S_i, O_1, \cdots, O_t|\lambda)$$

Viterbi Algorithm

$$s_t(i) = \max_{s_1 s_2 \cdots s_{t-1}} P(s_1, s_2 \cdots s_{t-1}, s_t = S_i, O_1, \cdots, O_t|\lambda)$$

- Initialization:

$$s_1(i) = \pi_i b_i(O_1)\,, \quad \psi_1(i) = 0$$

- Recursion:

$$\begin{aligned} s_t(j) &= \max_i s_{t-1}(i)a_{ij}b_j(O_t) \\ \psi_t(j) &= \arg\max_i s_{t-1}(i)a_{ij} \end{aligned}$$

- Termination:

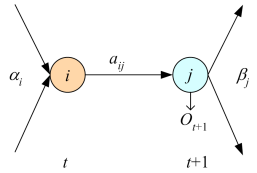
$$p^* = \max_i s_T(i)\,,\; s_T^* = \arg\max_i s_T(i)$$

- Backtracking:

$$s_t^* = \psi_{t+1}(s_{t+1}^*)\,,\; t = T-1, T-2, \ldots, 1$$

Parameter Learning

$$\begin{aligned} \xi_t(i,j) &= P(s_t = S_i, s_{t+1}|O,\lambda) \\ &= \frac{\alpha_t(i)a_{ij}b_j(O_{t+1})\beta_{t+1}(j)}{\sum_k\sum_\ell \alpha_t(k)a_{k\ell}b_\ell(O_{t+1})\beta_{t+1}(\ell)} \end{aligned}$$



- Use EM Algorithm:
  - Auxiliary variable:

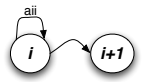
$$Z_i^t = \begin{cases} 1, & \text{if } s_t = S_i \\ 0, & \text{otherwise} \end{cases} \qquad Z_{ij}^t = \begin{cases} 1, & \text{if } s_t = S_i \text{ and } s_{t+1} = S_j \\ 0, & \text{otherwise} \end{cases}$$

  - E-Step: $E[z_i^t] = s_t(i) \quad E[z_{ij}^t] = \xi_t(i,j)$
    - recall: $s_t(i) = P(s_t = S_i|O,\lambda) = \frac{\alpha_t(i)\beta_t(i)}{\sum_{j=1}^{N}\alpha_t(j)\beta_t(j)}$
  - M-Step:

$$\begin{aligned} \hat{\pi}_i &= \frac{\sum_{k=1}^{K}}{s_t^k(i)}K \\ \hat{a}_{ij} &= \frac{\sum_{k=1}^{K}\sum_{t=1}^{T_k-1}\xi_t^k(i,j)}{\sum_{k=1}^{K}\sum_{t=1}^{T_k-1}s_t^k(i)} \\ \hat{b}_j(m) &= \frac{\sum_{k=1}^{K}\sum_{t=1}^{T_k-1}s_t^k(j)\delta(O_t^k - v_m)}{\sum_{k=1}^{K}\sum_{t=1}^{T_k-1}s_t^k(i)} \end{aligned}$$

HMM State Duration

- The time an observation would spend in a state is *implicit*:

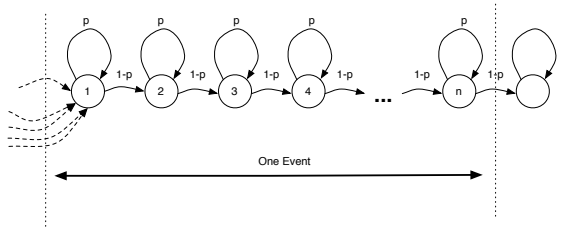


$$p_i(d) = a_{ii}^{(d-1)}(1 - a_{ii})$$

  - where $p_i(d)$ is the probability of staying 'd' discrete times in state *i*
  - This is an *exponential* model of time
  - Not very desirable for all temporal sequences... such as music!

Group Homework 4

- Consider the following Markov topology for one event:



1. What is the probability of staying 'd' times within the *n* states (or in the global event)?
2. What is the name of this probability model?!
3. Assuming *n* is fixed, solve for *p* for an expected duration *d*.

HMM Variants

- Discrete observations:

$$P(O_t \mid s_t = S_j, \lambda) = \prod_{m=1}^{M} b_j(m)^{r_m^t} \qquad r_m^t = \begin{cases} 1 & \text{if } O_t = v_m \\ 0 & \text{otherwise} \end{cases}$$

- Continuous observations:

$$P(O_t \mid s_t = S_j, \lambda) \sim \mathcal{N}(\mu_j, \sigma_j^2)$$

  use EM to find parameters... .

- Gaussian Mixtures:

$$P(O_t \mid q_t = S_j, \lambda) = \sum_{l=1}^{L} P(\mathcal{G}_{jl}) p(O_t \mid q_t = S_j, \mathcal{G}_l, \lambda) \sim \mathcal{N}(\mu_l, \Sigma_l)$$

- Duration-Focused models
  - Transitions as explicit functions of time... .
- And many more... .

HMMs

- So far, we have assumed that our underlying *models* are *static*!
  - In some cases this works out well as an approximation
  - In many cases it won't!
  - Real-life systems are *dynamic systems*
- Dynamics systems
  - Are generally hard to model... .
  - Much easier if they are linear... .
  - Much harder if they are non-linear... .

Kalman Filter Models

- Also known as *Linear Dynamical Systems*
- Imagine a sequential framework where latent variable *S* and observations *Y* are continuous
- and underlying dynamics is linear... . (or can be approximated so)
- Example:
  - A one dimensional tracking system:

$$\mathbf{s}_k = \begin{pmatrix} \text{position} \\ \text{velocity} \end{pmatrix}_k = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \mathbf{s}_{k-1} = \mathbf{A}\mathbf{s}_{k-1}$$

$$y_k = \text{position}_k = \begin{pmatrix} 1 & 0 \end{pmatrix} \mathbf{s}_k = \mathbf{C}\mathbf{s}_k$$

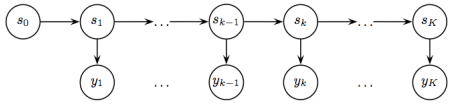
Kalman Filter Models

- Tracking example:
  - Imagine that we have unknown accelerations (dynamics!)

$$\begin{aligned} \mathbf{s}_k &= \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \mathbf{s}_{k-1} + \epsilon_k \\ &= \mathbf{A}\mathbf{s}_{k-1} + \epsilon_k \end{aligned}$$

$$\begin{aligned} y_k &= \begin{pmatrix} 1 & 0 \end{pmatrix} \mathbf{s}_k + \nu_k \\ &= \mathbf{C}\mathbf{s}_k + \nu_k \end{aligned}$$

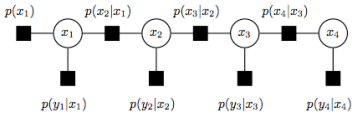
  - Generatively speaking,



$$\begin{aligned} \mathbf{s}_k &\sim \mathcal{N}(\mathbf{s}_k; \mathbf{A}\mathbf{s}_{k-1}, Q) \\ y_k &\sim \mathcal{N}(y_k; \mathbf{C}\mathbf{s}_k, R) \end{aligned}$$

Kalman Filter Models

Inference is similar to HMM: summations are replaced by integrations



- Forward Pass

$$p(y_{1:K}) = \underbrace{\int_{x_K} p(y_T|x_K) \underbrace{\int_{x_{K-1}} p(x_K|x_{K-1}) \cdots \int_{x_2} p(x_3|x_2) \underbrace{p(y_2|x_2) \overbrace{\int_{x_1} p(x_2|x_1)}^{\alpha_{2|1}} \underbrace{p(y_1|x_1) \overbrace{p(x_1)}^{\alpha_{1|0}}}_{\alpha_1}}_{\alpha_2}}_{\alpha_K}$$

- Backward Pass

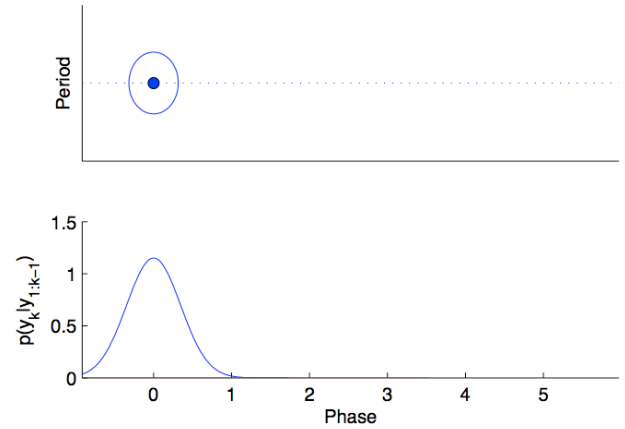
$$p(y_{1:K}) = \int_{x_1} p(x_1)p(y_1|x_1) \cdots \underbrace{\int_{x_{K-1}} p(x_{K-1}|x_{K-2})p(y_{K-1}|x_{K-1})}_{\beta_{K-2}} \underbrace{\int_{x_K} p(x_K|x_{K-1})p(y_K|x_K) \underbrace{1}_{\beta_K}}_{\beta_{K-1}}$$

Kalman Filter Models

- Goal:
  - Find an a posteriori belief based on prior estimate and a weighted difference between the *actual measurement* y_t and a *measurement prediction* $\hat{s}_t$
  - A series of *predictions* <> *corrections*
- *Results into closed form solutions in the Gaussian case...*

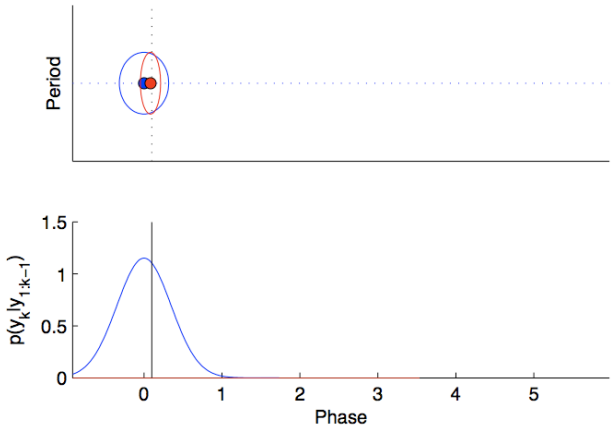
Kalman Filters

$p(s_1)$



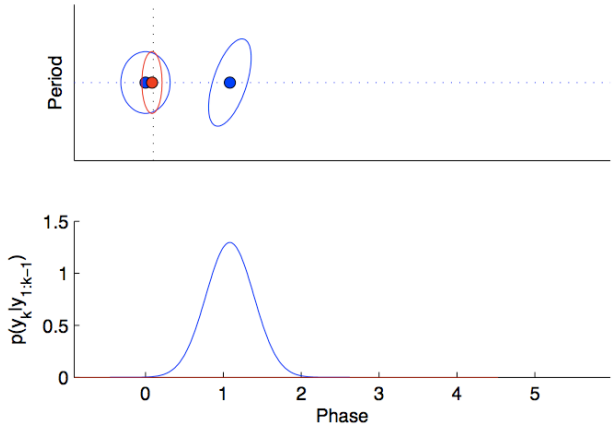
Kalman Filters

$$p(y_1|s_1)p(s_1)$$



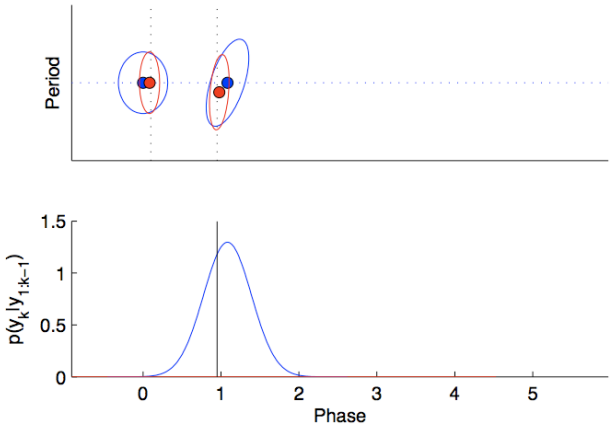
Kalman Filters

$$p(s_2|y_1) \propto \int ds_1 p(s_2|s_1)p(y_1|s_1)p(s_1)$$



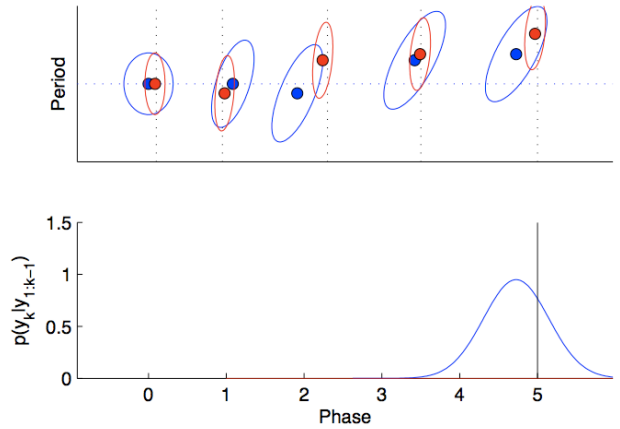
Kalman Filters

$$p(y_2|s_2)p(s_2|y_1)$$



Kalman Filters

$$p(s_5|y_{1:5})$$



Sequential learning applications in audio

- Score Following (AKA real-time alignment of audio to symbolic scores)
- Gesture Following
- Speech Recognition
- Automatic Transcription
- and many many many more... .