Review:

Probability and Statistics

Sam Roweis

Machine Learning Summer School, January 2005

Random Variables and Densities

- Random variables X represents outcomes or states of world.
  Instantiations of variables usually in lower case: x
  We will write $p(x)$ to mean probability$(X = x)$.
- Sample Space: the space of all possible outcomes/states.
  (May be discrete or continuous or mixed.)
- Probability mass (density) function $p(x) \geq 0$
  Assigns a non-negative number to each point in sample space.
  Sums (integrates) to unity: $\sum_x p(x) = 1$ or $\int_x p(x)dx = 1$.
  Intuitively: how often does x occur, how much do we believe in x.
- Ensemble: random variable + sample space+ probability function

Probability

- We use probabilities $p(x)$ to represent our beliefs $B(x)$ about the states x of the world.
- There is a formal calculus for manipulating uncertainties represented by probabilities.
- Any consistent set of beliefs obeying the *Cox Axioms* can be mapped into probabilities.
  1. Rationally ordered degrees of belief:
     if $B(x) > B(y)$ and $B(y) > B(z)$ then $B(x) > B(z)$
  2. Belief in x and its negation $\bar{x}$ are related: $B(x) = f[B(\bar{x})]$
  3. Belief in conjunction depends only on conditionals:
     $B(x \; and \; y) = g[B(x), B(y|x)] = g[B(y), B(x|y)]$

Expectations, Moments

- Expectation of a function $a(x)$ is written $E[a]$ or $\langle a \rangle$

$$E[a] = \langle a \rangle = \sum_x p(x)a(x)$$

  e.g. mean $= \sum_x xp(x)$, variance $= \sum_x (x - E[x])^2 p(x)$
- Moments are expectations of higher order powers.
  (Mean is first moment. Autocorrelation is second moment.)
- Centralized moments have lower moments subtracted away
  (e.g. variance, skew, curtosis).
- Deep fact: Knowledge of all orders of moments
  completely defines the entire distribution.

Means, Variances and Covariances

- Remember the definition of the mean and covariance of a vector random variable:

$$E[x] = \int_{\mathbf{x}} \mathbf{x} p(\mathbf{x}) d\mathbf{x} = \mathbf{m}$$

$$\mathrm{Cov}[x] = E[(\mathbf{x} - \mathbf{m})(\mathbf{x} - \mathbf{m})^\top] = \int_x (\mathbf{x} - \mathbf{m})(\mathbf{x} - \mathbf{m})^\top p(\mathbf{x}) d\mathbf{x} = \mathbf{V}$$

which is the expected value of the outer product of the variable with itself, after subtracting the mean.

- Also, the covariance between two variables:

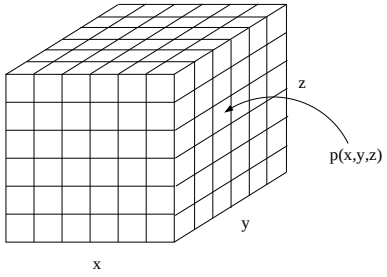
$$\mathrm{Cov}[\mathbf{x}, \mathbf{y}] = E[(\mathbf{x} - \mathbf{m_x})(\mathbf{y} - \mathbf{m_y})^\top] = \mathbf{C}$$
$$= \int_{\mathbf{xy}} (\mathbf{x} - \mathbf{m_x})(\mathbf{y} - \mathbf{m_y})^\top p(\mathbf{x}, \mathbf{y}) d\mathbf{x} d\mathbf{y} = \mathbf{C}$$

which is the expected value of the outer product of one variable with another, after subtracting their means.
Note: $\mathbf{C}$ is not symmetric.

Joint Probability

- Key concept: two or more random variables may interact. Thus, the probability of one taking on a certain value depends on which value(s) the others are taking.
- We call this a joint ensemble and write

$$p(x, y) = \text{prob}(X = x \text{ and } Y = y)$$

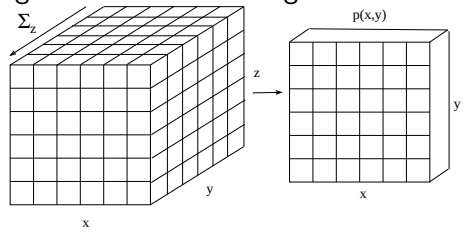


Marginal Probabilities

- We can "sum out" part of a joint distribution to get the *marginal distribution* of a subset of variables:

$$p(x) = \sum_y p(x, y)$$

- This is like adding slices of the table together.

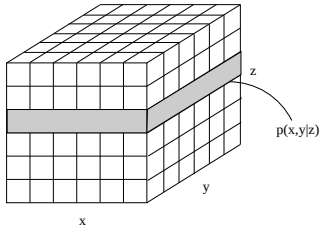


- Another equivalent definition: $p(x) = \sum_y p(x|y)p(y)$.

Conditional Probability

- If we know that some event has occurred, it changes our belief about the probability of other events.
- This is like taking a "slice" through the joint table.

$$\boxed{p(x|y) = p(x, y)/p(y)}$$



Bayes' Rule

- Manipulating the basic definition of conditional probability gives one of the most important formulas in probability theory:

$$\boxed{p(x|y) = \frac{p(y|x)p(x)}{p(y)} = \frac{p(y|x)p(x)}{\sum_{x'} p(y|x')p(x')}}$$

- This gives us a way of "reversing"conditional probabilities.
- Thus, all joint probabilities can be factored by selecting an ordering for the random variables and using the "chain rule":

$$\boxed{p(x, y, z, \ldots) = p(x)p(y|x)p(z|x, y)p(\ldots|x, y, z)}$$

Entropy

- Measures the amount of ambiguity or uncertainty in a distribution:

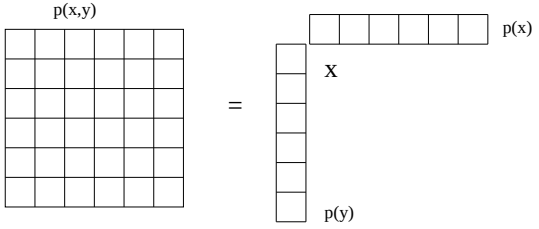
$$\boxed{H(p) = -\sum_x p(x) \log p(x)}$$

- Expected value of $-\log p(x)$ (a function which depends on p(x)!).
- $H(p) > 0$ unless only one possible outcomein which case $H(p) = 0$.
- Maximal value when p is uniform.
- Tells you the expected "cost" if each event costs $-\log p$(event)

Independence & Conditional Independence

- Two variables are independent iff their joint factors:

$$p(x, y) = p(x)p(y)$$



- Two variables are conditionally independent given a third one if for all values of the conditioning variable, the resulting slice factors:

$$p(x, y|z) = p(x|z)p(y|z) \qquad \forall z$$

Cross Entropy (KL Divergence)

- An assymetric measure of the distancebetween two distributions:

$$KL[p\|q] = \sum_x p(x)[\log p(x) - \log q(x)]$$

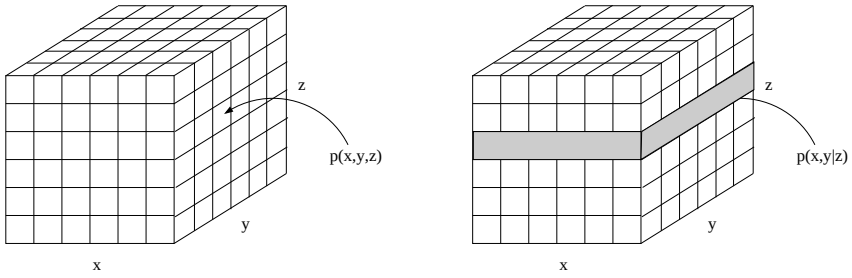
- $KL > 0$ unless $p = q$ then $KL = 0$
- Tells you the extra cost if events were generated by $p(x)$ but instead of charging under $p(x)$ you charged under $q(x)$.

Statistics

- Probability: inferring probabilistic quantities for data given fixed models (e.g. prob. of events, marginals, conditionals, etc).
- Statistics: inferring a model given fixed data observations (e.g. clustering, classification, regression).
- Many approaches to statistics:
  frequentist, *Bayesian*, *decision theory*, ...

(Conditional) Probability Tables

- For discrete (categorical) quantities, the most basic parametrization is the probability table which lists $p(x_i = k^{th} \text{ value})$.
- Since PTs must be nonnegative and sum to 1, for k-ary variables there are $k - 1$ free parameters.
- If a discrete variable is conditioned on the values of some other discrete variables we make one table for each possible setting of the parents: these are called *conditional probability tables* or CPTs.



Some (Conditional) Probability Functions

- Probability density functions $p(x)$ (for continuous variables) or probability mass functions $p(x = k)$ (for discrete variables) tell us how likely it is to get a particular value for a random variable (possibly conditioned on the values of some other variables.)
- We can consider various types of variables: binary/discrete (categorical), continuous, interval, and integer counts.
- For each type we'll see some basic *probability models* which are parametrized families of distributions.

Exponential Family

- For (continuous or discrete) random variable $\mathbf{x}$

$$p(\mathbf{x}|\eta) = h(\mathbf{x}) \exp\{\eta^\top T(\mathbf{x}) - A(\eta)\}$$
$$= \frac{1}{Z(\eta)} h(\mathbf{x}) \exp\{\eta^\top T(\mathbf{x})\}$$

  is an exponential family distribution with *natural parameter* η.
- Function $T(\mathbf{x})$ is a *sufficient statistic*.
- Function $A(\eta) = \log Z(\eta)$ is the log normalizer.
- Key idea: all you need to know about the data is captured in the summarizing function $T(\mathbf{x})$.

Bernoulli Distribution

- For a binary random variable $x = \{0, 1\}$ with $p(x = 1) = \pi$:

$$\begin{aligned} p(x|\pi) &= \pi^x (1-\pi)^{1-x} \\ &= \exp\left\{\log\left(\frac{\pi}{1-\pi}\right) x + \log(1-\pi)\right\} \end{aligned}$$

- Exponential family with:

$$\begin{aligned} \eta &= \log\frac{\pi}{1-\pi} \\ T(x) &= x \\ A(\eta) &= -\log(1-\pi) = \log(1+e^{\eta}) \\ h(x) &= 1 \end{aligned}$$

- The *logistic* function links natural parameter and chance of heads

$$\pi = \frac{1}{1+e^{-\eta}} = \text{logistic}(\eta)$$

Multinomial

- For a categorical (discrete), random variable taking on K possible values, let π_k be the probability of the k^{th} value. We can use a binary vector $\mathbf{x} = (x_1, x_2, \ldots, x_k, \ldots, x_K)$ in which $x_k{=}1$ if and only if the variable takes on its k^{th} value. Now we can write,

$$p(\mathbf{x}|\pi) = \pi_1^{x_1}\pi_2^{x_2}\cdots\pi_K^{x_K} = \exp\left\{\sum_i x_i \log \pi_i\right\}$$

  Exactly like a probability table, but written using binary vectors.

- If we observe this variable several times $\mathbf{X} = \{\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^N\}$, the (iid) probability depends on the *total observed counts* of each value:

$$p(\mathbf{X}|\pi) = \prod_n p(\mathbf{x}^n|\pi) = \exp\left\{\sum_i \left(\sum_n x_i^n\right) \log \pi_i\right\} = \exp\left\{\sum_i c_i \log \pi_i\right\}$$

Poisson

- For an integer count variable with *rate* λ:

$$\begin{aligned} p(x|\lambda) &= \frac{\lambda^x e^{-\lambda}}{x!} \\ &= \frac{1}{x!}\exp\{x\log\lambda - \lambda\} \end{aligned}$$

- Exponential family with:

$$\begin{aligned} \eta &= \log\lambda \\ T(x) &= x \\ A(\eta) &= \lambda = e^{\eta} \\ h(x) &= \frac{1}{x!} \end{aligned}$$

- e.g. number of photons $\mathbf{x}$ that arrive at a pixel during a fixed interval given mean intensity λ
- Other count densities: (neg)binomial, geometric.

Multinomial as Exponential Family

- The multinomial parameters are constrained: $\sum_i \pi_i = 1$.
  Define (the last) one in terms of the rest: $\pi_K = 1 - \sum_{i=1}^{K-1} \pi_i$

$$p(\mathbf{x}|\pi) = \exp\left\{\sum_{i=1}^{K-1} \log\left(\frac{\pi_i}{\pi_K}\right) x_i + k\log\pi_K\right\}$$

- Exponential family with:

$$\begin{aligned} \eta_i &= \log\pi_i - \log\pi_K \\ T(x_i) &= x_i \\ A(\eta) &= -k\log\pi_K = k\log\textstyle\sum_i e^{\eta_i} \\ h(\mathbf{x}) &= 1 \end{aligned}$$

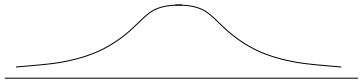
- The *softmax* function relates direct and natural parameters:

$$\pi_i = \frac{e^{\eta_i}}{\sum_j e^{\eta_j}}$$

Gaussian (normal)

- For a continuous univariate random variable:

$$p(x|\mu,\sigma^2) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left\{-\frac{1}{2\sigma^2}(x-\mu)^2\right\}$$
$$= \frac{1}{\sqrt{2\pi}\sigma}\exp\left\{\frac{\mu x}{\sigma^2} - \frac{x^2}{2\sigma^2} - \frac{\mu^2}{2\sigma^2} - \log\sigma\right\}$$



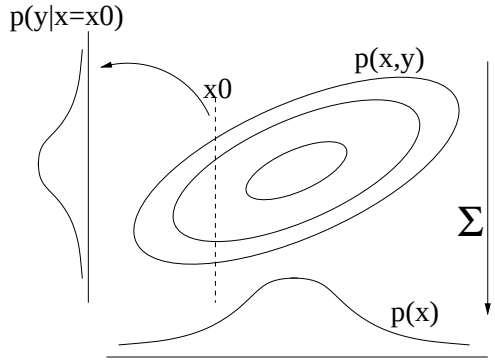
- Exponential family with:

$$\eta = [\mu/\sigma^2\,;\,-1/2\sigma^2]$$
$$T(x) = [x\,;\,x^2]$$
$$A(\eta) = \log\sigma + \mu/2\sigma^2$$
$$h(x) = 1/\sqrt{2\pi}$$

- Note: a univariate Gaussian is a two-parameter distribution with a two-component vector of sufficient statistis.

Important Gaussian Facts

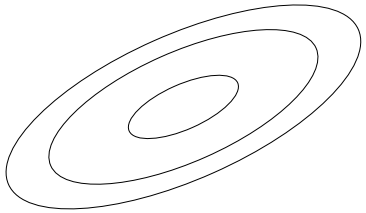
- All marginals of a Gaussian are again Gaussian.
  Any conditional of a Gaussian is again Gaussian.



Multivariate Gaussian Distribution

- For a continuous vector random variable:

$$p(x|\mu,\Sigma) = |2\pi\Sigma|^{-1/2}\exp\left\{-\frac{1}{2}(\mathbf{x}-\mu)^\top\Sigma^{-1}(\mathbf{x}-\mu)\right\}$$



- Exponential family with:

$$\eta = [\Sigma^{-1}\mu\,;\,-1/2\Sigma^{-1}]$$
$$T(x) = [\mathbf{x}\,;\,\mathbf{x}\mathbf{x}^\top]$$
$$A(\eta) = \log|\Sigma|/2 + \mu^\top\Sigma^{-1}\mu/2$$
$$h(x) = (2\pi)^{-n/2}$$

- Sufficient statistics: mean vector and correlation matrix.
- Other densities: Student-t, Laplacian.
- For non-negative values use exponential, Gamma, log-normal.

Gaussian Marginals/Conditionals

- To find these parameters is mostly linear algebra:
  Let $\mathbf{z} = [\mathbf{x}^\top\mathbf{y}^\top]^\top$ be normally distributed according to:

$$\mathbf{z} = \begin{bmatrix}\mathbf{x}\\ \mathbf{y}\end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix}\mathbf{a}\\ \mathbf{b}\end{bmatrix}; \begin{bmatrix}\mathbf{A} & \mathbf{C}\\ \mathbf{C}^\top & \mathbf{B}\end{bmatrix}\right)$$

  where $\mathbf{C}$ is the (non-symmetric) cross-covariance matrix between $\mathbf{x}$ and $\mathbf{y}$ which has as many rows as the size of $\mathbf{x}$ and as many columns as the size of $\mathbf{y}$.
  The marginal distributions are:

$$\mathbf{x} \sim \mathcal{N}(\mathbf{a};\mathbf{A})$$
$$\mathbf{y} \sim \mathcal{N}(\mathbf{b};\mathbf{B})$$

  and the conditional distributions are:

$$\mathbf{x}|\mathbf{y} \sim \mathcal{N}(\mathbf{a}+\mathbf{C}\mathbf{B}^{-1}(\mathbf{y}-\mathbf{b});\mathbf{A}-\mathbf{C}\mathbf{B}^{-1}\mathbf{C}^\top)$$
$$\mathbf{y}|\mathbf{x} \sim \mathcal{N}(\mathbf{b}+\mathbf{C}^\top\mathbf{A}^{-1}(\mathbf{x}-\mathbf{a});\mathbf{B}-\mathbf{C}^\top\mathbf{A}^{-1}\mathbf{C})$$

Parameter Constraints

- If we want to use general optimizations (e.g. conjugate gradient) to learn latent variable models, we often have to make sure parameters respect certain constraints. (e.g. $\sum_k \alpha_k = 1$, Σ_k pos.definite).
- A good trick is to reparameterize these quantities in terms of unconstrained values. For mixing proportions, use the softmax:

$$\alpha_k = \frac{\exp(q_k)}{\sum_j \exp(q_j)}$$

- For covariance matrices, use the Cholesky decomposition:

$$\Sigma^{-1} = A^\top A$$
$$|\Sigma|^{-1/2} = \prod_i A_{ii}$$

where A is upper diagonal with positive diagonal:

$$A_{ii} = \exp(r_i) > 0 \qquad A_{ij} = a_{ij} \quad (j > i) \qquad A_{ij} = 0 \quad (j < i)$$

Parameterizing Conditionals

- When the variable(s) being conditioned on (parents) are discrete, we just have one density for each possible setting of the parents. e.g. a table of natural parameters in exponential models or a table of tables for discrete models.
- When the conditioned variable is continuous, its value sets some of the parameters for the other variables.
- A very common instance of this for regression is the "linear-Gaussian": $p(\mathbf{y}|\mathbf{x}) = \text{gauss}(\theta^\top \mathbf{x}; \Sigma)$.
- For discrete children and continuous parents, we often use a Bernoulli/multinomial whose paramters are some function $f(\theta^\top \mathbf{x})$.

Moments

- For continuous variables, moment calculations are important.
- We can easily compute moments of any exponential family distribution by taking the derivatives of the log normalizer $A(\eta)$.
- The q^{th} derivative gives the q^{th} centred moment.

$$\frac{dA(\eta)}{d\eta} = \text{mean}$$
$$\frac{d^2A(\eta)}{d\eta^2} = \text{variance}$$
$$\cdots$$

- When the sufficient statistic is a vector, partial derivatives need to be considered.

Generalized Linear Models (GLMs)

- Generalized Linear Models: $p(\mathbf{y}|\mathbf{x})$ is exponential family with conditional mean $\mu = f(\theta^\top \mathbf{x})$.
- The function f is called the *response function*; if we chose it to be the inverse of the mapping b/w conditional mean and natural parameters then it is called the *canonical response function*.

$$\eta = \psi(\mu)$$
$$f(\cdot) = \psi^{-1}(\cdot)$$

- We can be even more general and define distributions by arbitrary *energy* functions proportional to the log probability.

$$p(\mathbf{x}) \propto \exp\{-\sum_k H_k(\mathbf{x})\}$$

- A common choice is to use pairwise terms in the energy:

$$H(\mathbf{x}) = \sum_i a_i x_i + \sum_{\text{pairs } ij} w_{ij} x_i x_j$$

Matrix Inversion Lemma (Sherman-Morrison-Woodbury Formulae)

- There is a good trick for inverting matrices when they can be decomposed into the sum of an easily inverted matrix (D) and a low rank outer product. It is called the *matrix inversion lemma*.

$$(D - AB^{-1}A^\top)^{-1} = D^{-1} + D^{-1}A(B - A^\top D^{-1}A)^{-1}A^\top D^{-1}$$

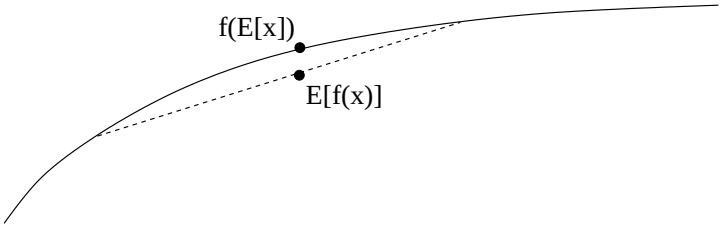
- The same trick can be used to compute determinants:

$$\log|D + AB^{-1}A^\top| = \log|D| - \log|B| + \log|B + A^\top D^{-1}A|$$

Jensen's Inequality

- For any concave function $f()$ and any distribution on x,

$$\boxed{E[f(x)] \leq f(E[x])}$$



- e.g. $\log()$ and $\sqrt{\ }$ are concave
- This allows us to bound expressions like $\log p(x) = \log \sum_z p(x, z)$

Matrix Derivatives

- Here are some useful matrix derivatives:

$$\frac{\partial}{\partial A}\log|A| = (A^{-1})^\top$$

$$\frac{\partial}{\partial A}\text{trace}[B^\top A] = B$$

$$\frac{\partial}{\partial A}\text{trace}[BA^\top CA] = 2CAB$$

Logsum

- Often you can easily compute $b_k = \log p(\mathbf{x}|z = k, \theta_k)$, but it will be very negative, say -10^6 or smaller.
- Now, to compute $\ell = \log p(\mathbf{x}|\theta)$ you need to compute $\log \sum_k e^{b_k}$. (e.g. for calculating responsibilities at test time or for learning)
- Careful! Do not compute this by doing `log(sum(exp(b)))`. You will get underflow and an incorrect answer.
- Instead do this:
  - Add a constant exponent B to all the values b_k such that the largest value comes close to the maxiumum exponent allowed by machine precision: `B = MAXEXPONENT-log(K)-max(b)`.
  - Compute `log(sum(exp(b+B)))-B`.
- Example: if $\log p(x|z = 1) = -120$ and $\log p(x|z = 2) = -120$, what is $\log p(x) = \log [p(x|z = 1) + p(x|z = 2)]$?
  Answer: $\log[2e^{-120}] = -120 + \log 2$.

Lecture 1:

Probabilistic Graphical Models

Sam Roweis

Monday January 24, 2005
Machine Learning Summer School

Core vs. Probabilistic AI

- KR: work with facts/assertions; develop rules of logical inference
- Planning: work with applicability/effects of actions; develop searches for actions which achieve goals/avert disasters.
- Expert Systems: develop by hand a set of rules for examining inputs, updating internal states and generating outputs
- Learning approach: use probabilistic models to tune performance based on many data examples.
- Probabilistic AI: emphasis on noisy measurements, approximation in hard cases, learning, algorithmic issues.
  logical assertions $\Rightarrow$ probability distributions
  logical inference $\Rightarrow$ conditional probability distributions
  logical operators $\Rightarrow$ probabilistic generative models

Intelligent Computers

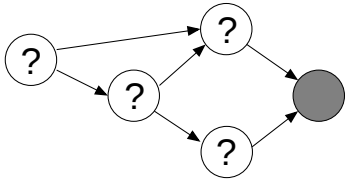
- We want intelligent, adaptive, robust behaviour.


$\Rightarrow$ Sam Roweis

- Often hand programming not possible.
- Solution? Get the computer to program itself, by showing it examples of the behaviour we want!
  This is the *learning* approach to AI.
- Really, we write the structure of the program and the computer tunes many internal parameters.

The Power of Learning

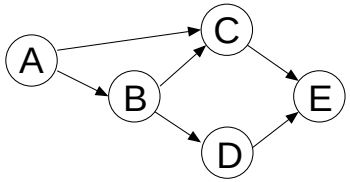
- Probabilistic Databases
  - traditional DB technology cannot answer queries about items that were never loaded into the dataset
  - UAI models are like probabilistic databases



- Automatic System Building
  - old expert systems needed hand coding of knowledge and of output semantics
  - learning automatically constructs rules and supports all types of queries

Uncertainty and Artificial Intelligence (UAI)

- Probabilistic methods can be used to:
  - make decisions given partial information about the world
  - account for noisy sensors or actuators
  - explain phenomena not part of our models
  - describe inherently stochastic behaviour in the world



- Example: you live in California with your spouse and two kids. You listen to the radio on your dirve home, and when you arrive you find your burglar alarm ringing.
  Do you think your house was broken into?

Applications of Probabilistic Learning

- Automatic speech recognition & speaker verification
- Printed and handwritten text parsing
- Face location and identification
- Tracking/separating objects in video
- Search and recommendation (e.g. `google`, `amazon`)
- Financial prediction, fraud detection (e.g. credit cards)
- Insurance premium prediction, product pricing
- Medical diagnosis/image analysis (e.g. pneumonia, pap smears)
- Game playing (e.g. backgammon)
- Scientific analysis/data visualization (e.g. galaxy classification)
- Analysis/control of complex systems (e.g. freeway traffic, industrial manufacturing plants, space shuttle)
- Troubleshooting and fault correction

Other Names for UAI

- Machine learning, data mining, applied statistics, adaptive (stochastic) signal processing, probabilistic planning/reasoning...
- Some differences:
  - Data mining almost always uses large data sets, statistics almost always small ones.
  - Data mining, planning, decision theory often have no internal parameters to be learned.
  - Statistics often has no algorithm to run!
  - ML/UAI algorithms are rarely online and rarely scale to huge data (changing now).
- Learning is most useful when the structure of the task is not well understood but can be characterized by a dataset with strong statistical regularity. Also useful in adaptive or dynamic situations when the task (or its parameters) are constantly changing.

Related Areas of Study

- Adaptive data compression/coding:
  – state of the art methods for image compression and error correcting codes all use learning methods
- Stochastic signal processing:
  – denoising, source separation, scene analysis, morphing
- Decision making, planning:
  – use both utility and uncertainty optimally, e.g. influence diagrams
- Adaptive software agents / auctions / preferences – action choice under limited resources and reward signals

Canonical Tasks

- *Supervised Learning*: given examples of inputs and corresponding desired outputs, predict outputs on future inputs.
  Ex: classification, regression, time series prediction
- *Unsupervised Learning*: given only inputs, automatically discover representations, features, structure, etc.
  Ex: clustering, outlier detection, compression
- *Rule Learning*: given multiple measurements, discover very common joint settings of subsets of measurements.
- *Reinforcement Learning*: given sequences of inputs, actions from a fixed set, and scalar rewards/punishments, learn to select action sequences in a way that maximizes expected reward.
  [Last two will not be covered in these lectures.]

Using random variables to represent the world

- We will use mathematical random variables to encode everything we know about the task: inputs, outputs and internal states.
- Random variables may be *discrete/categorical* or *continuous/vector*.
  Discrete quantities take on one of a fixed set of values,
  e.g. `{0,1}`, `{email,spam}`, `{sunny,overcast,raining}`.
  Continuous quantities take on real values.
  e.g. `temp=12.2, income=38231, blood-pressure=58.9`
- Generally have repeated measurements of same quantities.
  Convention: $i, j, \ldots$ indexes components/variables/dimensions;
  $n, m, \ldots$ indexes cases/records, x are inputs, y are outputs.
  – x_i^n is the value of the i^{th} input variable on the n^{th} case
  – y_j^m is the value of the j^{th} output variable on the m^{th} case
  $\mathbf{x}_n$ is a vector of all inputs for the n^{th} case
  $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n, \ldots, \mathbf{x}_N\}$ are all the inputs

Representation

- Key issue: how do we represent information about the world? (e.g. for an image, do we just list pixel values in some order?)

 → 127,254,3,18,...

- We must pick a way of numerically representing things that exploits regularities or structure in the data.
- To do this, we will rely on probability and statistics, and in particular on *random variables*.
- A *random variable* is like a variable in a computer program that represents a certain quantity, but its value changes depending on which data our program is looking at. The value a random variables is often unknown/uncertain, so we use probabilities.

Structure of Learning Machines

- Given some inputs, expressed in our representation, how do we calculate something about them (e.g. this is Sam's face)?
- Our computer program uses a *mathematical function* $z = f(x)$
  x is the representation of our input (e.g. face)
  z is the representation of our output (e.g. Sam)
- Hypothesis Space and Parameters:
  We don't just make up functions out of thin air. We select them from a carefully specified set, known as our *hypothesis space*.
- Generally this space is indexed by a set of *parameters* θ which are knobs we can turn to create different machines:
  $\mathcal{H} : \{f(\mathbf{z}|\mathbf{x}, \theta)\}$
- Hardest part of doing probabilistic learning is deciding how to represent inputs/outputs and how to select hypothesis spaces.

Loss Functions for Tuning Parameters

- Let inputs=$\mathbf{X}$, correct answers=$\mathbf{Y}$, outputs of our machine=$\mathbf{Z}$.
- Once we select a representation and hypothesis space, how do we set our parameters θ?
- We need to quantify what it means to do well or poorly on a task.
- We can do this by defining a loss function $L(\mathbf{X}, \mathbf{Y}, \mathbf{Z})$ (or just $L(\mathbf{X}, \mathbf{Z})$ in unsupervised case).
- Examples:
  Classification: $z_n(\mathbf{x}_n)$ is predicted class. $L = \sum_n [y_n \neq z_n(\mathbf{x}_n)]$

  Regression: $\mathbf{z}_n(\mathbf{x}_n)$ is predicted output. $L = \sum_n \|\mathbf{y}_n - \mathbf{z}_n(\mathbf{x}_n)\|^2$

  Clustering: $\mathbf{z}_c$ is mean of all cases assigned to cluster c. $L = \sum_n \min_c \|\mathbf{x}_n - \mathbf{z}_c\|^2$
- Now set parameters to minimize average loss function.

Sampling Assumption

- Imagine that our data is created randomly, from a joint probability distribution $p(\mathbf{x}, \mathbf{y})$ which we don't know.
- We are given a finite (possibly noisy) training sample: $\{\mathbf{x}_1, \mathbf{y}_1, \ldots, \mathbf{x}_n, \mathbf{y}_n, \ldots, \mathbf{x}_N \mathbf{y}_N\}$ with members n generated *independently and identically distributed* (iid).
- Looking only at the training data, we construct a machine that generates outputs $\mathbf{z}$ given inputs. (Possibly by trying to build machines with small training error.)
- Now a new sample is drawn from the same distribution as the training sample.
- We run our machine on the new sample and evaluate the loss; this is the test error.
- Central question: by looking at the machine, the training data and the training error, what if anything can be said about test error?

Training vs. Testing

- *Training data*: the $\mathbf{X}, \mathbf{Y}$ we are given.
  Testing data: the $\mathbf{X}, \mathbf{Y}$ we will see in future.
- *Training error*: the average value of loss on the training data.
  Test error: the average value of loss on the test data.
- What is our real goal? To do well on the data we have seen already? Usually not. We already have the answers for that data. We want to perform well on *future unseen data*. So ideally we would like to minimize the test error. How to do this if we don't have test data?
- Probabilistic framework to the rescue!

Generalization and Overfitting

- Crucial concepts: *generalization*, *capacity*, *overfitting*.
- What's the danger in the above setup? That we will do well on training data but poorly on test data. This is called *overfitting*.
- Example: just memorize training data and give random outputs on all other data.
- Key idea: you can't learn anything about the world without making some assumptions.
  (Although you can memorize what you have seen).
- Both representation and hypothesis class (model choice) represent assumptions we make.
- The ability to achieve small loss on test data is *generalization*.

Capacity: Complexity of Hypothesis Space

- Learning == Search in Hypothesis Space
- Inductive Learning Hypothesis: Generalization is possible. If a machine performs well on most training data AND it is not too complex, it will probably do well on similar test data.
- Amazing fact: in many cases this can actually be *proven*. In other words, if our hypothesis space is not too complicated/flexible (has a low *capacity* in some formal sense), and if our training set is large enough then we can bound the probability of performing much worse on test data than on training data.
- The above statement is carefully formalized in 20 years of research in the area of *learning theory*.

Formal Setup

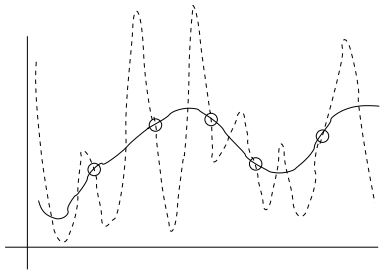
- Cast machine learning tasks as *numerical optimization problems*.
- Quantify how well the machine pleases us by a scalar *objective function* which we can evaluate on sets of inputs/outputs.
- Represent given inputs/outputs as arguments to this function.
- Also introduce a set of unknown parameters θ which are also arguments of the objective function.
- Goal: adjust unknown parameters to minimize objective function given inputs/outputs.

$$\arg\min_{\theta} \Phi(\mathbf{X}, \mathbf{Y}|\theta)$$

- The *art* of designing a machine learning system is to select the numerical representation of the inputs/outputs and the mathematical formulation of the task as an objective function.
- The *mechanics* involve optimizing the objective function given the observed data to find the best parameters. (Often leads to art!)

Inductive Bias

- The converse of the Inductive Learning Hypothesis is that generalization only possible if we make some assumptions, or introduce some priors. We need an *Inductive Bias*.
- No Free Lunch Theorems: an unbiased learner can never generalize.
- Consider: arbitrarily wiggly functions or random truth tables or non-smooth distributions.



0 0 0	0
0 0 1	?
0 1 0	1
0 1 1	1
1 0 0	0
1 0 1	?
1 1 0	1
1 1 1	?

General Objective Functions

- The general structure of the objective function is:

$$\Phi(\mathbf{X}, \theta) = L(\mathbf{X}|\theta) + P(\theta)$$

- L is the loss function, and P is a penalty function which penalizes more complex models.
- This says that it is good to fit the data well (get low training loss) but it is also good to bias ourselves towards simpler models to avoid overfitting.

Probabilistic Approach

- Given the above setup, we can think of learning as estimation of joint probability density functions given samples from the functions.
- Classification and Regression: conditional density estimation $p(\mathbf{y}|\mathbf{x})$
- Unsupervised Learning: density estimation $p(\mathbf{x})$
- *The central object of interest is the joint distribution and the main difficulty is compactly representing it and robustly learning its shape given noisy samples.*
- *Our inductive bias is expresses as prior assumptions about these joint distributions.*
- *The main computations we will need to do during the operation of our algorithms are to efficiently calculate marginal and conditional distributions from our compactly represented joint model.*

Conditional Independence

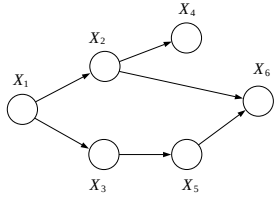
- Notation: $\mathbf{x}_A \perp \mathbf{x}_B|\mathbf{x}_C$
  Definition: two (sets of) variables $\mathbf{x}_A$ and $\mathbf{x}_B$ are conditionally independent given a third $\mathbf{x}_C$ if:
  $$\mathsf{P}(\mathbf{x}_A, \mathbf{x}_B|\mathbf{x}_C) = \mathsf{P}(\mathbf{x}_A|\mathbf{x}_C)\mathsf{P}(\mathbf{x}_B|\mathbf{x}_C) \qquad \forall \mathbf{x}_C$$
  which is equivalent to saying
  $$\mathsf{P}(\mathbf{x}_A|\mathbf{x}_B, \mathbf{x}_C) = \mathsf{P}(\mathbf{x}_A|\mathbf{x}_C) \qquad \forall \mathbf{x}_C$$
- Only a subset of all distributions respect any given (nontrivial) conditional independence statement. The subset of distributions that respect all the CI assumptions we make is the *family of distributions consisitent with our assumptions*.
- Probabilistic graphical models are a powerful, elegant and simple way to specify such a family.

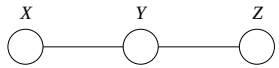
Joint Probabilities

- Goal 1: represent a joint distribution $\mathsf{P}(\mathbf{X}) = \mathsf{P}(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n)$ compactly even when there are many variables.
- Goal 2: efficiently calculate marginal and conditionals of such compactly represented joint distributions.
- Notice: for n discrete variables of arity k, the naive (table) representation is HUGE: it requires k^n entries.
- We need to make some *assumptions* about the distribution.
  One simple assumption: independence $==$ complete factorization:
  $$\mathsf{P}(\mathbf{X}) = \prod_i \mathsf{P}(\mathbf{x}_i)$$
- But the independence assumption is too restrictive.
  So we make *conditional independence* assumptions instead.

Probabilistic Graphical Models

- Probabilistic graphical models represent large joint distributions compactly using a set of "local" relationships specified by a graph.
- Each random variable in our model corresponds to a graph node.
- There are directed/undirected *edges* between the nodes which tell us qualitatively about the *factorization* of the joint probability.
- There are *functions* stored at the nodes which tell us the quantitative details of the pieces into which the distribution factors.





- Graphical models are also known as Bayes(ian) (Belief) Net(work)s.

Directed Graphical Models

- Consider *directed acyclic graphs* over n variables.
- Each node has (possibly empty) set of parents π_i.
- Each node maintains a function $f_i(x_i; \mathbf{x}_{\pi_i})$ such that $f_i > 0$ and $\sum_{x_i} f_i(x_i; \mathbf{x}_{\pi_i}) = 1 \, \forall \pi_i$.
- Define the joint probability to be:

$$\mathsf{P}(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n) = \prod_i f_i(x_i; \mathbf{x}_{\pi_i})$$

  Even with no further restriction on the the f_i, it is always true that

$$f_i(x_i; \mathbf{x}_{\pi_i}) = \mathsf{P}(x_i | \mathbf{x}_{\pi_i})$$

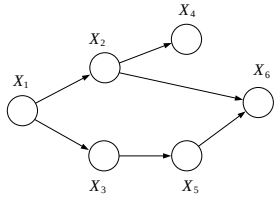
  so we will just write

$$\mathsf{P}(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n) = \prod_i \mathsf{P}(x_i | \mathbf{x}_{\pi_i})$$

- Factorization of the joint in terms of *local conditional probabilities*. Exponential in "fan-in" of each node instead of in total variables n.

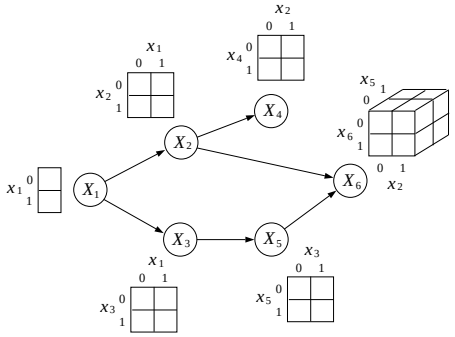
Example DAG

- Consider this six node network:



The joint probability is now:

$$\mathsf{P}(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5, \mathbf{x}_6) = \mathsf{P}(\mathbf{x}_1)\mathsf{P}(\mathbf{x}_2|\mathbf{x}_1)\mathsf{P}(\mathbf{x}_3|\mathbf{x}_1)\mathsf{P}(\mathbf{x}_4|\mathbf{x}_2)\mathsf{P}(\mathbf{x}_5|\mathbf{x}_3)\mathsf{P}(\mathbf{x}_6|\mathbf{x}_2, \mathbf{x}_5)$$



Conditional Independence in DAGs

- If we order the nodes in a directed graphical model so that parents always come before their children in the ordering then the graphical model implies the following about the distribution:

$$\{x_i \perp \mathbf{x}_{\widetilde{\pi}_i} | \mathbf{x}_{\pi_i}\} \forall i$$

  where $\mathbf{x}_{\widetilde{\pi}_i}$ are the nodes coming before x_i that are not its parents.
- In other words, the DAG is telling us that each variable is conditionally independent of its non-descendants given its parents.
- Such an ordering is called a "topological" ordering.

Missing Edges

- Key point about directed graphical models:
  Missing edges imply conditional independence
- Remember, that by the chain rule we can always write the full joint as a product of conditionals, given an ordering:

$$\mathsf{P}(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \ldots) = \mathsf{P}(\mathbf{x}_1)\mathsf{P}(\mathbf{x}_2|\mathbf{x}_1)\mathsf{P}(\mathbf{x}_3|\mathbf{x}_1, \mathbf{x}_2)\mathsf{P}(\mathbf{x}_4|\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3)\ldots$$

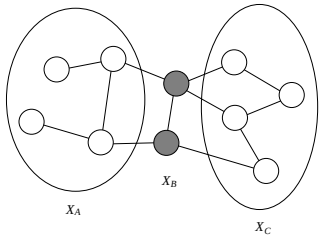
- If the joint is represented by a DAGM, then some of the conditioned variables on the right hand sides are missing. This is equivalent to enforcing conditional independence.
- Start with the "idiot's graph": each node has all previous nodes in the ordering as its parents.
- Now remove edges to get your DAG.
- Removing an edge into node i eliminates an argument from the conditional probability factor $p(x_i|\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{i-1})$

Even more structure

- Surprisingly, once you have specified the basic conditional independencies, there are other ones that follow from those.
- In general, it is a hard problem to say which extra CI statements follow from a basic set. However, in the case of DAGMs, we have an efficient way of generating *all* CI statements that *must be true* given the connectivity of the graph.
- This involves the idea of *d-separation* in a graph.
- Notice that for specific (numerical) choices of factors at the nodes there may be even more conditional independencies, but we are only concerned with statements that are always true of every member of the family of distributions, no matter what specific factors live at the nodes.
- Remember: the graph alone represents a *family of joint distributions consistent with its CI assumptions*, not any specific distribution.

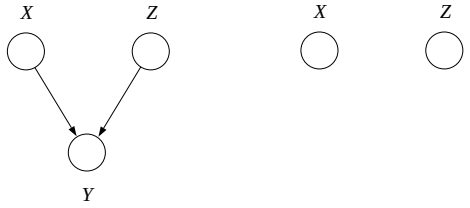
Undirected Models

- Also graphs with one node per random variable and edges that connect pairs of nodes, but now the edges are undirected.
- Semantics: every node is conditionally independent from its non-neighbours given its neighbours, i.e.
  $\mathbf{x}_A \perp \mathbf{x}_C \mid \mathbf{x}_B$ if every path b/w $\mathbf{x}_A$ and $\mathbf{x}_C$ goes through $\mathbf{x}_B$



- Can model symmetric interactions that directed models cannot.
- aka Markov Random Fields, Markov Networks, Boltzmann Machines, Spin Glasses, Ising Models

Explaining Away



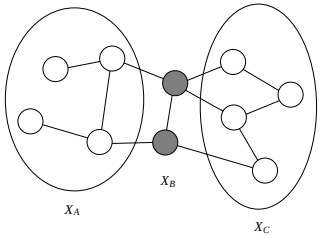
- Q: When we condition on $\mathbf{y}$, are $\mathbf{x}$ and $\mathbf{z}$ independent?

$$\mathrm{P}(\mathbf{x}, \mathbf{y}, \mathbf{z}) = \mathrm{P}(\mathbf{x})\mathrm{P}(\mathbf{z})\mathrm{P}(\mathbf{y}|\mathbf{x}, \mathbf{z})$$

- $\mathbf{x}$ and $\mathbf{z}$ are *marginally independent*, but given $\mathbf{y}$ they are *conditionally dependent*.
- This important effect is called *explaining away* (Berkson's paradox.)
- For example, flip two coins independently; let $\mathbf{x}$=coin1,$\mathbf{z}$=coin2. Let $\mathbf{y}$=1 if the coins come up the same and $\mathbf{y}$=0 if different.
- $\mathbf{x}$ and $\mathbf{z}$ are independent, but if I tell you $\mathbf{y}$, they become coupled!

Simple Graph Separation

- In undirected models, simple graph separation (as opposed to d-separation) tells us about conditional independencies.
- $\mathbf{x}_A \perp \mathbf{x}_C|\mathbf{x}_B$ if every path between $\mathbf{x}_A$ and $\mathbf{x}_C$ is blocked by some node in $\mathbf{x}_B$.



- "Markov Ball" algorithm:
  remove $\mathbf{x}_B$ and see if there is any path from $\mathbf{x}_A$ to $\mathbf{x}_C$.

Conditional Parameterization?

- In directed models, we started with $p(\mathbf{X}) = \prod_i p(x_i|\mathbf{x}_{\pi_i})$ and we derived the d-separation semantics from that.
- Undirected models: have the semantics, need parametrization.
- What about this "conditional parameterization"?

$$p(\mathbf{X}) = \prod_i p(x_i|\mathbf{x}_{\text{neighbours}(i)})$$

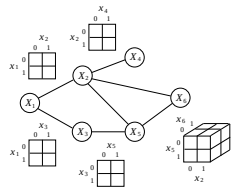
- Good: product of local functions.
  Good: each one has a simple conditional interpretation.
  Bad: local functions cannot be arbitrary, but must agree properly in order to define a valid distribution.

Clique Potentials

- Whatever factorization we pick, we know that only connected nodes can be arguments of a single local function.
- A *clique* is a fully connected subset of nodes.
- Thus, consider using a *product of positive clique potentials*:

$$\mathsf{P}(\mathbf{X}) = \frac{1}{Z} \prod_{\text{cliques } c} \psi_c(\mathbf{x}_c) \qquad Z = \sum_{\mathbf{X}} \prod_{\text{cliques } c} \psi_c(\mathbf{x}_c)$$

- The product of functions that don't need to agree with each other.
- Still factors in the way that the graph semantics demand.
- Without loss of generality we can restrict ourselves to *maximal cliques*. (Why?)



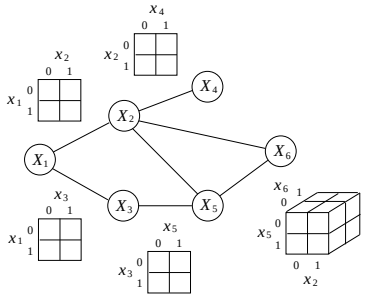
Marginal Parameterization?

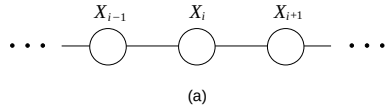
- OK, what about this "marginal parameterization"?

$$p(\mathbf{X}) = \prod_i p(x_i, \mathbf{x}_{\text{neighbours}(i)})$$

- Good: product of local functions.
  Good: each one has a simple marginal interpretation.
  Bad: only very few pathalogical marginals on overalpping nodes can be multiplied to give a valid joint.

Examples of Clique Potentials





(a)

x_{i-1} \ x_i	−1	1
−1	1.5	0.2
1	0.2	1.5

x_i \ x_{i+1}	−1	1
−1	1.5	0.2
1	0.2	1.5

(b)

Boltzmann Distributions

- We often represent the clique potentials using their logs:

$$\psi_C(\mathbf{x}_C) = \exp\{-H_C(\mathbf{x}_C)\}$$

for arbitrary real valued "energy" functions $H_C(\mathbf{x}_C)$.
The negative sign is a standard convention.

- This gives the joint a nice additive structure:

$$\mathsf{P}(\mathbf{X}) = \frac{1}{Z}\exp\{-\sum_{\text{cliques } C} H_C(\mathbf{x}_c)\} = \frac{1}{Z}\exp\{-H(\mathbf{X})\}$$

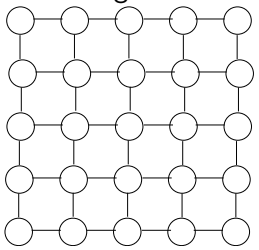
where the sum in the exponent is called the "free energy":

$$H(\mathbf{X}) = \sum_C H_C(\mathbf{x}_c)$$

- This way of defining a probability distribution based on energies is the "Boltzmann distribution" from statistical physics.

Example: Ising Models

- Common model for binary nodes: spin-glass/ Ising lattice.
- Nodes are arranged in a regular topology (often a regular packing grid) and connected only to their geometric neighbours.



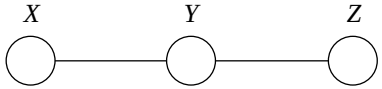
- For example, if we think of each node as a pixel, we might want to encourage nearby pixels to have similar intensities.
- Energy is of the form:

$$H(\mathbf{x}) = \sum_{ij} \beta_{ij} x_i x_j + \sum_i \alpha_i x_i$$

Partition Function

- Normalizer $Z(\mathbf{X})$ above is called the "partition function".
- Computing the normalizer and its derivatives can often be the hardest part of inferene and learning in undirected models.
- Often the factored structure of the distribution makes it possible to efficiently do the sums/integrals required to compute Z.
- Don't *always* have to compute Z, e.g. for conditional probabilities.

Interpretation of Clique Potentials



- The model implies $\mathbf{x} \perp \mathbf{z} \mid \mathbf{y}$

$$p(\mathbf{x}, \mathbf{y}, \mathbf{z}) = p(\mathbf{y})p(\mathbf{x}|\mathbf{y})p(\mathbf{z}|\mathbf{y})$$

- We can write this as:

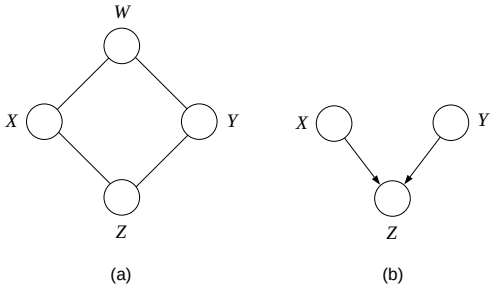
$$p(\mathbf{x}, \mathbf{y}, \mathbf{z}) = p(\mathbf{x}, \mathbf{y})p(\mathbf{z}|\mathbf{y}) = \psi_{\mathbf{xy}}(\mathbf{x}, \mathbf{y})\psi_{\mathbf{yz}}(\mathbf{y}, \mathbf{z})$$
$$p(\mathbf{x}, \mathbf{y}, \mathbf{z}) = p(\mathbf{x}|\mathbf{y})p(\mathbf{z}, \mathbf{y}) = \psi_{\mathbf{xy}}(\mathbf{x}, \mathbf{y})\psi_{\mathbf{yz}}(\mathbf{y}, \mathbf{z})$$

cannot have all potentials be marginals
cannot have all potentials be conditionals

- The positive clique potentials can only be thought of as general "compatibility", "goodness" or "happiness" functions over their variables, but not as probability distributions.

Expressive Power

- Can we always convert directed $\leftrightarrow$ undirected?
- No.

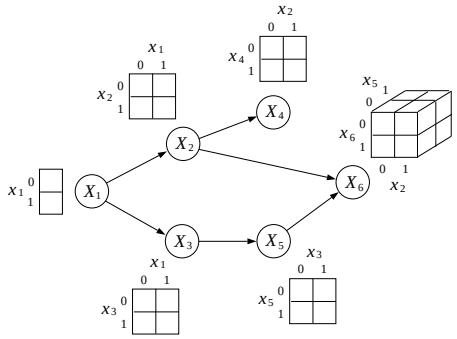


No directed model can represent these and only these independencies.
$\mathbf{x} \perp \mathbf{y} \mid \{\mathbf{w}, \mathbf{z}\}$
$\mathbf{w} \perp \mathbf{z} \mid \{\mathbf{x}, \mathbf{y}\}$

No undirected model can represent these and only these independencies.
$\mathbf{x} \perp \mathbf{y}$

Probability Tables & CPTs

- For discrete (categorical) variables, the most basic parametrization is the probability table which lists $p(x = k^{th} \text{ value})$.
- Since PTs must be nonnegative and sum to 1, for k-ary nodes there are $k - 1$ free parameters.
- If a discrete node has discrete parent(s) we make one table for each setting of the parents: this is a *conditional probability table* or CPT.



What's Inside the Nodes/Cliques?

- We've focused a lot on the structure of the graphs in directed and undirected models. Now we'll look at specific functions that can live inside the nodes (directed) or on the cliques (undirected).
- For directed models we need prior functions $p(x_i)$ for root nodes and parent-conditionals $p(x_i|\mathbf{x}_{\pi_i})$ for interior nodes.
- For undirected models we need clique potentials $\psi_C(\mathbf{x}_C)$ on the maximal cliques (or log potentials/energies $H_C(\mathbf{x}_C)$).
- We'll consider various types of nodes: binary/discrete (categorical), continuous, interval, and integer counts.
- We'll see some basic *probability models* (parametrized families of distributions); these models live inside nodes of directed models.
- We'll also see a variety of potential/energy functions which take multiple node values as arguments and return a scalar compatibility; these live on the cliques of undirected models.

Exponential Family

- For a numeric random variable $\mathbf{x}$

$$p(\mathbf{x}|\eta) = h(\mathbf{x}) \exp\{\eta^\top T(\mathbf{x}) - A(\eta)\}$$
$$= \frac{1}{Z(\eta)} h(\mathbf{x}) \exp\{\eta^\top T(\mathbf{x})\}$$

  is an exponential family distribution with *natural parameter* η.
- Function $T(\mathbf{x})$ is a *sufficient statistic*.
- Function $A(\eta) = \log Z(\eta)$ is the log normalizer.
- Key idea: all you need to know about the data in order to estimate parameters is captured in the summarizing function $T(\mathbf{x})$.
- Examples: Bernoulli, binomial/geometric/negative-binomial, Poisson, gamma, multinomial, Gaussian, ...

Moments

- For numeric nodes, moment calculations are important.
- We can easily compute moments of any exponential family distribution by taking the derivatives of the log normalizer $A(\eta)$.
- The q^{th} derivative gives the q^{th} centred moment.

$$\frac{dA(\eta)}{d\eta} = \text{mean}$$
$$\frac{d^2A(\eta)}{d\eta^2} = \text{variance}$$
$$\ldots$$

- When the sufficient statistic is a vector, partial derivatives need to be considered.

GLMs and Canonical Links

- Generalized Linear Models: $p(\mathbf{y}|\mathbf{x})$ is exponential family with conditional mean $\mu_i = f_i(\theta^\top \mathbf{x})$.
- The function f is called the *response function*.
- If we chose f to be the inverse of the mapping b/w conditional mean and natural parameters then it is called the *canonical response function* or *canonical link*:

$$\eta = \psi(\mu)$$
$$f(\cdot) = \psi^{-1}(\cdot)$$

- Example: logistic function is canonical link for Bernoulli variables; softmax function is canonical link for multinomials

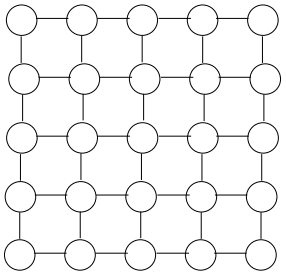
Nodes with Parents

- When the parent is discrete, we just have one probability model for each setting of the parent. Examples:
  – table of natural parameters (exponential model for cts. child)
  – table of tables (CPT model for discrete child)
- When the parent is numeric, some or all of the parameters for the child node become *functions* of the parent's value.
- A very common instance of this for regression is the "linear-Gaussian": $p(\mathbf{y}|\mathbf{x}) = \text{gauss}(\theta^\top \mathbf{x}; \Sigma)$.
- For classification, often use Bernoulli/Multinomial densities whose parameters π are some function of the parent: $\pi_j = f_j(\mathbf{x})$.

Potential Functions

- We are much less constrained with potential functions, since they can be any positive function of the values of the clique nodes.
- Recall $\psi_C(\mathbf{x}_C) = \exp\{-H_C(\mathbf{x}_C)\}$
- A common (redundant) choice for cliques which are pairs is:

$$H(\mathbf{x}) = \sum_i a_i x_i + \sum_{\text{pairs } ij} w_{ij} x_i x_j$$



Lecture 2:

Parameter Learning in Fully Observed Graphical Models

Sam Roweis

Tuesday January 25, 2005
Machine Learning Summer School

Learning Graphical Models from Data

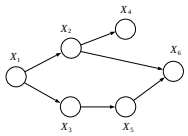
- In AI the bottleneck is often knowledge acquisition.
- Human experts are rare, expensive, unreliable, slow.
  But we have lots of machine readable data.
- Want to build systems automatically based on data and a small amount of prior information (e.g. from experts).

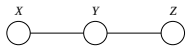
 ⇒ Sam Roweis  ⇒ Geoff Hinton

- In this course, our "systems" will be probabilistic graphical models.
- Assume the prior information we have specifies type & structure of the GM, as well as the mathematical form of the parent-conditional distributions or clique potentials.
- In this case learning ≡ setting parameters.
  ("Structure learning" is also possible but we won't consider it now.)

Review: Goal of Graphical Models

- Graphical models aim to provide *compact factorizations* of large joint probability distributions.
- These factorizations are achieved using *local functions* which exploit *conditional independencies* in the models.
- The graph tells us a basic set of *conditional independencies* that must be true. From these we can derive more that also must be true. These independencies are crucial to developing efficient algorithms valid for *all* numerical settings of the local functions.
- Local functions tell us the quantitative details of the distribution.
- Certain numerical settings of the distribution may have more independencies present, but these do not come from the *graph*.





Basic Statistical Problems

- Let's remind ourselves of the basic problems we discussed on the first day: *density estimation*, *clustering classification* and *regression*.
- Can always do joint density estimation and then condition:
  Regression: $p(\mathbf{y}|\mathbf{x}) = p(\mathbf{y},\mathbf{x})/p(\mathbf{x}) = p(\mathbf{y},\mathbf{x})/\int p(\mathbf{y},\mathbf{x})d\mathbf{y}$
  Classification: $p(c|\mathbf{x}) = p(c,\mathbf{x})/p(\mathbf{x}) = p(c,\mathbf{x})/\sum_c p(c,\mathbf{x})$
  Clustering: $p(c|\mathbf{x}) = p(c,\mathbf{x})/p(\mathbf{x})$ c unobserved
  Density Estimation: $p(\mathbf{y}|\mathbf{x}) = p(\mathbf{y},\mathbf{x})/p(\mathbf{x})$ $\mathbf{x}$ unobserved

In general, if certain nodes are *always* observed we may not want to model their density:

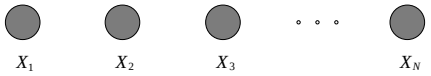
Regression/Classification

If certain nodes are *always* unobserved they are called *hidden* or *latent* variables (more later):

Clustering/Density Est.

Multiple Observations, Complete Data, IID Sampling

- A single observation of the data $\mathbf{X}$ is rarely useful on its own.
- Generally we have data including many observations, which creates a *set of random variables*: $\mathcal{D} = \{\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^M\}$
- We will assume two things (for now):
  1. Observations are independently and identically distributed according to joint distribution of graphical model: IID samples.
  2. We observe all random variables in the domain on each observation: complete data.
- We shade the nodes in a graphical model to indicate they are observed. (Later you will see unshaded nodes corresponding to missing data or latent variables.)



Maximum Likelihood

- For IID data:

$$p(\mathcal{D}|\theta) = \prod_m p(\mathbf{x}^m|\theta)$$
$$\ell(\theta; \mathcal{D}) = \sum_m \log p(\mathbf{x}^m|\theta)$$

- Idea of maximum likelihod estimation (MLE): pick the setting of parameters most likely to have generated the data we saw:

$$\theta^*_{\mathrm{ML}} = \mathrm{argmax}_\theta \ \ell(\theta; \mathcal{D})$$

- Very commonly used in statistics.
  Often leads to "intuitive", "appealing", or "natural" estimators.
- For a start, the IID assumption makes the log likelihood into a sum, so its derivative can be easily taken term by term.

Likelihood Function

- So far we have focused on the (log) probability function $p(\mathbf{x}|\theta)$ which assigns a probability (density) to any joint configuration of variables $\mathbf{x}$ given fixed parameters θ.
- But in learning we turn this on its head: we have some fixed data and we want to find parameters.
- Think of $p(\mathbf{x}|\theta)$ as a function of θ for fixed $\mathbf{x}$:

$$L(\theta; \mathbf{x}) = p(\mathbf{x}|\theta)$$
$$\ell(\theta; \mathbf{x}) = \log p(\mathbf{x}|\theta)$$

  This function is called the (log) "likelihood".
- Chose θ to maximize some cost function $c(\theta)$ which includes $\ell(\theta)$:

$c(\theta) = \ell(\theta; \mathcal{D})$ maximum likelihood (ML)
$c(\theta) = \ell(\theta; \mathcal{D}) + r(\theta)$ maximum a posteriori (MAP)/penalizedML

(also cross-validation, Bayesian estimators, BIC, AIC, ...)

Example: Bernoulli Trials

- We observe M iid coin flips: $\mathcal{D}$=H,H,T,H,...
- Model: $p(H) = \theta \quad p(T) = (1 - \theta)$
- Likelihood:

$$\begin{aligned}
\ell(\theta; \mathcal{D}) &= \log p(\mathcal{D}|\theta) \\
&= \log \prod_m \theta^{\mathbf{x}^m} (1-\theta)^{1-\mathbf{x}^m} \\
&= \log\theta \sum_m \mathbf{x}^m + \log(1-\theta) \sum_m (1 - \mathbf{x}^m) \\
&= \log\theta N_{\mathrm{H}} + \log(1-\theta) N_{\mathrm{T}}
\end{aligned}$$

- Take derivatives and set to zero:

$$\frac{\partial \ell}{\partial \theta} = \frac{N_{\mathrm{H}}}{\theta} - \frac{N_{\mathrm{T}}}{1-\theta}$$
$$\Rightarrow \theta^*_{\mathrm{ML}} = \frac{N_{\mathrm{H}}}{N_{\mathrm{H}} + N_{\mathrm{T}}}$$

Example: Multinomial

- We observe M iid die rolls (K-sided): $\mathcal{D}$=3,1,K,2,...
- Model: $p(k) = \theta_k \quad \sum_k \theta_k = 1$
- Likelihood (for binary indicators $[\mathbf{x}^m = k]$):

$$\begin{aligned}\ell(\theta; \mathcal{D}) &= \log p(\mathcal{D}|\theta) \\ &= \log \prod_m \theta_{\mathbf{x}^m} = \log \prod_m \theta_1^{[\mathbf{x}^m=1]} \dots \theta_k^{[\mathbf{x}^m=k]} \\ &= \sum_k \log \theta_k \sum_m [\mathbf{x}^m = k] = \sum_k N_k \log \theta_k\end{aligned}$$

- Take derivatives and set to zero (enforcing $\sum_k \theta_k = 1$):

$$\begin{aligned}\frac{\partial \ell}{\partial \theta_k} &= \frac{N_k}{\theta_k} - M \\ \Rightarrow \theta_k^* &= \frac{N_k}{M}\end{aligned}$$

Example: Univariate Normal

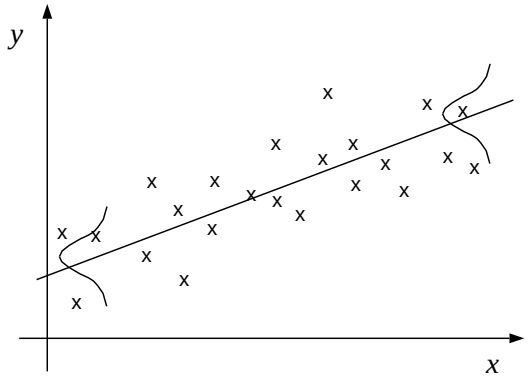
- We observe M iid real samples: $\mathcal{D}$=1.18,-.25,.78,...
- Model: $p(x) = (2\pi\sigma^2)^{-1/2} \exp\{-(x-\mu)^2/2\sigma^2\}$
- Likelihood (using probability density):

$$\begin{aligned}\ell(\theta; \mathcal{D}) &= \log p(\mathcal{D}|\theta) \\ &= -\frac{M}{2}\log(2\pi\sigma^2) - \frac{1}{2}\sum_m \frac{(x^m - \mu)^2}{\sigma^2}\end{aligned}$$

- Take derivatives and set to zero:

$$\begin{aligned}\frac{\partial \ell}{\partial \mu} &= (1/\sigma^2)\textstyle\sum_m (x_m - \mu) \\ \frac{\partial \ell}{\partial \sigma^2} &= -\frac{M}{2\sigma^2} + \frac{1}{2\sigma^4}\textstyle\sum_m (x_m - \mu)^2 \\ \Rightarrow \mu_{\mathrm{ML}} &= (1/M)\textstyle\sum_m x_m \\ \sigma^2_{\mathrm{ML}} &= (1/M)\textstyle\sum_m x_m^2 - \mu^2_{\mathrm{ML}}\end{aligned}$$

Example: Linear Regression



Example: Linear Regression

- At a linear regression node, some parents (covariates/inputs) and all children (responses/outputs) are continuous valued variables.
- For each child and setting of discrete parents we use the model:

$$p(y|\mathbf{x}, \theta) = \mathrm{gauss}(y|\theta^\top \mathbf{x}, \sigma^2)$$

- The likelihood is the familiar "squared error" cost:

$$\ell(\theta; \mathcal{D}) = -\frac{1}{2\sigma^2}\sum_m (y^m - \theta^\top \mathbf{x}^m)^2$$

- The ML parameters can be solved for using linear least-squares:

$$\begin{aligned}\frac{\partial \ell}{\partial \theta} &= -\sum_m (y^m - \theta^\top \mathbf{x}^m)\mathbf{x}^m \\ \Rightarrow \theta^*_{\mathrm{ML}} &= (\mathbf{X}^\top \mathbf{X})^{-1}\mathbf{X}^\top \mathbf{Y}\end{aligned}$$

- Sufficient statistics are input correlation matrix and input-output cross-correlation vector.

Sufficient Statistics

- A statistic is a (possibly vector valued) function of a (set of) random variable(s).
- $T(\mathbf{X})$ is a "sufficient statistic" for $\mathbf{X}$ if

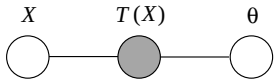
$$T(\mathbf{x}^1) = T(\mathbf{x}^2) \quad \Rightarrow L(\theta; \mathbf{x}^1) = L(\theta; \mathbf{x}^2) \quad \forall\theta$$

- Equivalently (by the Neyman factorization theorem) we can write:

$$p(\mathbf{x}|\theta) = h(\mathbf{x}, T(\mathbf{x}))\, g(T(\mathbf{x}), \theta)$$

- Example: exponential family models:

$$p(\mathbf{x}|\theta) = h(\mathbf{x}) \exp\{\eta^\top T(\mathbf{x}) - A(\eta)\}$$



(b)

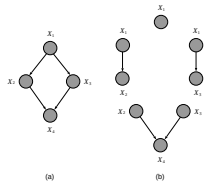
(c)

MLE for Directed GMs

- For a directed GM, the likelihood function has a nice form:

$$\log p(\mathcal{D}|\theta) = \log \prod_m \prod_i p(\mathbf{x}_i^m|\mathbf{x}_{\pi_i}, \theta_i) = \sum_m \sum_i \log p(\mathbf{x}_i^m|\mathbf{x}_{\pi_i}, \theta_i)$$

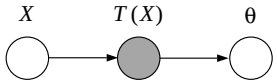
- The parameters *decouple*; so we can maximize likelihood independently for each node's function by setting θ_i.
- Only need the values of x_i and its parents in order to estimate θ_i.
- Furthermore, if $x_i, \mathbf{x}_{\pi_i}$ have sufficient statistics only need those.
- In general, for fully observed data if we know how to estimate params at a single node we can do it for the whole network.



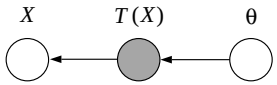
(a) (b)

Sufficient Statistics are Sums

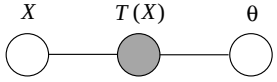
- In the examples above, the sufficient statistics were merely sums (counts) of the data:
  Bernoulli: # of heads, tails
  Multinomial: # of each type
  Gaussian: mean, mean-square
  Regression: correlations
- As we will see, this is true for all exponential family models: sufficient statistics are the average natural parameters.
- Only* exponential family models have simple sufficient statistics.



(a)



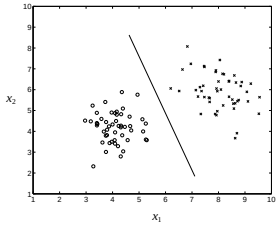
(b)



(c)

Reminder: Classification

- Given examples of a discrete *class label* y and some *features* $\mathbf{x}$.
- Goal: compute label (y) for new inputs $\mathbf{x}$.
- Two approaches:
  Generative: model $p(\mathbf{x}, y) = p(y)p(\mathbf{x}|y)$;
  use Bayes' rule to infer conditional $p(y|\mathbf{x})$.
  Discriminative: model discriminants $f(y|\mathbf{x})$ directly and take max.
- Generative approach is related to conditional *density estimation* while discriminative approach is closer to *regression*.



Probabilistic Classification: Bayes Classifiers

- Generative model: $p(\mathbf{x}, y) = p(y)p(\mathbf{x}|y)$.
  $p(y)$ are called class *priors*.
  $p(\mathbf{x}|y)$ are called *class conditional feature distributions*.
- For the prior we use a Bernoulli or multinomial:
  $p(y = k|\pi) = \pi_k$ with $\sum_k \pi_k = 1$.
- Classification rules:
  ML: $\operatorname{argmax}_y p(\mathbf{x}|y)$ (can behave badly if skewed priors)
  MAP: $\operatorname{argmax}_y p(y|\mathbf{x}) = \operatorname{argmax}_y \log p(\mathbf{x}|y) + \log p(y)$ (safer)
- Fitting: maximize $\sum_n \log p(\mathbf{x}^n, y^n) = \sum_n \log p(\mathbf{x}^n|y^n) + \log p(y^n)$
  1) Sort data into batches by class label.
  2) Estimate $p(y)$ by counting size of batches (plus regularization).
  3) Estimate $p(\mathbf{x}|y)$ separately within each batch using ML.
  (also with regularization).

Gaussian Class-Conditional Distributions

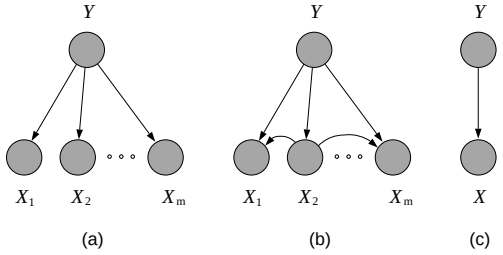
- If all features are continuous, a popular choice is a Gaussian class-conditional.

$$p(\mathbf{x}|y = k, \theta) = |2\pi\Sigma|^{-1/2} \exp\left\{-\frac{1}{2}(\mathbf{x} - \mu_k)\Sigma^{-1}(\mathbf{x} - \mu_k)\right\}$$

- Fitting: use the following amazing and useful fact.
  The maximum likelihood fit of a Gaussian to some data is the Gaussian whose mean is equal to the data mean and whose covariance is equal to the sample covariance.
  [Try to prove this as an exercise in understanding likelihood, algebra, and calculus all at once!]
- Seems easy. And works amazingly well.
  But we can do even better with some simple regularization...

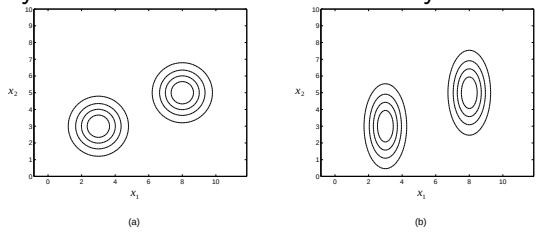
Three Key Regularization Ideas

- To avoid overfitting, we can put *priors* on the parameters of the class and class conditional feature distributions.
- We can also *tie* some parameters together so that fewer of them are estimated using more data.
- Finally, we can make *factorization* or *independence* assumptions about the distributions. In particular, for the class conditional distributions we can assume the features are fully dependent, partly dependent, or independent (!).



Regularized Gaussians

- Idea 1: assume all the covariances are the same (tie parameters).
  This is exactly Fisher's linear discriminant analysis.



- Idea 2: Make independence assumptions to get diagonal or identity-multiple covariances. (Or sparse inverse covariances.)
  More on this in a few minutes...
- Idea 3: add a bit of the identity matrix to each sample covariance.
  This "fattens it up" in directions where there wasn't enough data.
  Equivalent to using a "Wishart prior" on the covariance matrix.

Gaussian Bayes Classifier

- Maximum likelihood estimates for parameters:
  priors π_k: use observed frequencies of classes (plus smoothing)
  means μ_k: use class means
  covariance Σ: use data from single class or pooled data
  $(\mathbf{x}^m - \mu_{y^m})$ to estimate full/diagonal covariances
- Compute the posterior via Bayes' rule:

$$\begin{aligned} p(y=k|\mathbf{x},\theta) &= \frac{p(\mathbf{x}|y=k,\theta)p(y=k|\pi)}{\sum_j p(\mathbf{x}|y=j,\theta)p(y=j|\pi)} \\ &= \frac{\exp\{\mu_k^\top\Sigma^{-1}\mathbf{x} - \mu_k^\top\Sigma^{-1}\mu_k/2 + \log\pi_k\}}{\sum_j \exp\{\mu_j^\top\Sigma^{-1}\mathbf{x} - \mu_j^\top\Sigma^{-1}\mu_j/2 + \log\pi_j\}} \\ &= e^{\beta_k^\top\mathbf{x}}/\sum_j e^{\beta_j^\top\mathbf{x}} = \exp\{\beta_k^\top\mathbf{x}\}/Z \end{aligned}$$

  where $\beta_k = [\Sigma^{-1}\mu_k\,;\,(\mu_k^\top\Sigma^{-1}\mu_k + \log\pi_k)]$ and we have augmented $\mathbf{x}$ with a constant component always equal to 1 (bias term).

Softmax/Logit

- The squashing function is known as the *softmax* or *logit*:

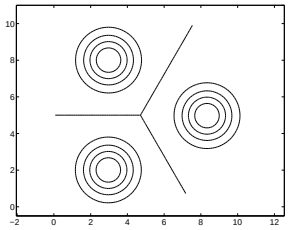
$$\phi_k(\mathbf{z}) \equiv \frac{e^{z_k}}{\sum_j e^{z_j}} \qquad g(\eta) = \frac{1}{1+e^{-\eta}}$$

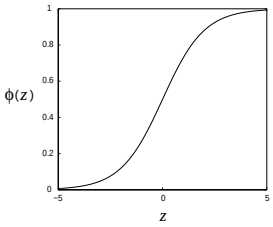
- It is invertible (up to a constant):

$$z_k = \log\phi_k + c \qquad \eta = \log(g/1-g)$$

- Derivative is easy:

$$\frac{\partial\phi_k}{\partial z_j} = \phi_k(\delta_{kj} - \phi_j) \qquad \frac{dg}{d\eta} = g(1-g)$$





Linear Geometry

- Taking the ratio of any two posteriors (the "odds") shows that the contours of equal pairwise probability are linear surfaces in the feature space:

$$\frac{p(y=k|\mathbf{x},\theta)}{p(y=j|\mathbf{x},\theta)} = \exp\left\{(\beta_k - \beta_j)^\top\mathbf{x}\right\}$$

- The pairwise discrimination contours $p(y_k) = p(y_j)$ are orthogonal to the differences of the means in feature space when $\Sigma = \sigma I$.
  For general Σ shared b/w all classes the same is true in the transformed feature space $\mathbf{w} = \Sigma^{-1}\mathbf{x}$.
- The priors do not change the geometry, they only shift the operating point on the logit by the log-odds $\log(\pi_k/\pi_j)$.
- Thus, for equal class-covariances, we obtain a *linear classifier*.
- If we use different covariances, the decision surfaces are conic sections and we have a quadratic classifier.

Exponential Family Class-Conditionals

- Bayes Classifier has the same softmax form whenever the class-conditional densities are *any* exponential family density:

$$\begin{aligned} p(\mathbf{x}|y=k,\eta_k) &= h(\mathbf{x})\exp\{\eta_k^\top\mathbf{x} - a(\eta_k)\} \\ p(y=k|\mathbf{x},\eta) &= \frac{p(\mathbf{x}|y=k,\eta_k)p(y=k|\pi)}{\sum_j p(\mathbf{x}|y=j,\eta_j)p(y=j|\pi)} \\ &= \frac{\exp\{\eta_k^\top\mathbf{x} - a(\eta_k)\}}{\sum_j \exp\{\eta_j^\top\mathbf{x} - a(\eta_j)\}} \\ &= \frac{e^{\beta_k^\top\mathbf{x}}}{\sum_j e^{\beta_j^\top\mathbf{x}}} \end{aligned}$$

  where $\beta_k = [\eta_k\,;\,-a(\eta_k)]$ and we have augmented $\mathbf{x}$ with a constant component always equal to 1 (bias term).
- Resulting classifier is linear in the sufficient statistics.

Discrete Bayesian Classifier

- If the inputs are discrete (categorical), what should we do?
- The simplest class conditional model is a joint multinomial (table):

$$p(x_1 = a, x_2 = b, \ldots | y = c) = \eta^c_{ab\ldots}$$

- This is conceptually correct, but there's a big practical problem.
- Fitting: ML params are observed counts:

$$\eta^c_{ab\ldots} = \frac{\sum_n [y_n = c][x_1 = a][x_2 = b][\ldots][\ldots]}{\sum_n [y_n = c]}$$

- Consider the 16x16 digits at 256 gray levels.
- How many entries in the table? How many will be zero? What happens at test time? Doh!
- We obviously need some regularlization. Smoothing will not help much here. Unless we know about the relationships between inputs beforehand, sharing parameters is hard also. But what about independence?

Discrete (Multinomial) Naive Bayes

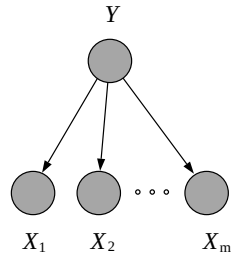
Discrete features x_i, assumed independent given the class label y.

$$p(x_i = j | y = k) = \eta_{ijk}$$
$$p(\mathbf{x} | y = k, \eta) = \prod_i \prod_j \eta_{ijk}^{[x_i = j]}$$

Classification rule:

$$p(y = k | \mathbf{x}, \eta) = \frac{\pi_k \prod_i \prod_j \eta_{ijk}^{[x_i = j]}}{\sum_q \pi_q \prod_i \prod_j \eta_{ijq}^{[x_i = j]}} = \frac{e^{\beta_k^\top \mathbf{x}}}{\sum_q e^{\beta_q^\top \mathbf{x}}}$$

$\beta_k = \log[\eta_{11k} \ldots \eta_{1jk} \ldots \eta_{ijk} \ldots \log \pi_k]$
$\mathbf{x} = [x_1{=}1; x_1{=}2; \ldots; x_i{=}j; \ldots; 1]$



(a)

Naive (Idiot's) Bayes Classifier

- Assumption: conditioned on class, attributes are independent.

$$p(\mathbf{x}|y) = \prod_i p(x_i|y)$$

- Sounds crazy right? Right! But it works.
- Algorithm: sort data cases into bins according to y_n. Compute marginal probabilities $p(y = c)$ using frequencies.
- For each class, estimate distribution of i^{th} variable: $p(x_i|y = c)$.
- At test time, compute $\text{argmax}_c\, p(c|\mathbf{x})$ using

$$c(\mathbf{x}) = \text{argmax}_c\, p(c|\mathbf{x}) = \text{argmax}_c\, [\log p(\mathbf{x}|c) + \log p(c)]$$
$$= \text{argmax}_c\, [\log p(c) + \sum_i \log p(x_i|c)]$$

Fitting Discrete Naive Bayes

- ML parameters are class-conditional frequency counts:

$$\eta^*_{ijk} = \frac{\sum_m [x_i{}^m = j][y^m = k]}{\sum_m [y^m = k]}$$

- How do we know? Write down the likelihood:

$$\ell(\theta; \mathcal{D}) = \sum_m \log p(y^m|\pi) + \sum_{mi} \log p(x_i{}^m | y^m, \eta)$$

and optimize it by setting its derivative to zero (careful! enforce normalization with Lagrange multipliers):

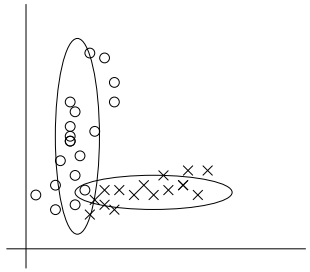
$$\ell(\eta; \mathcal{D}) = \sum_m \sum_{ijk} [x_i{}^m = j][y^m = k] \log \eta_{ijk} + \sum_{ik} \lambda_{ik}(1 - \textstyle\sum_j \eta_{ijk})$$

$$\frac{\partial \ell}{\partial \eta_{ijk}} = \frac{\sum_m [x_i{}^m = j][y^m = k]}{\eta_{ijk}} - \lambda_{ik}$$

$$\frac{\partial \ell}{\partial \eta_{ijk}} = 0 \quad \Rightarrow \lambda_{ik} = \sum_m [y^m = k] \quad \Rightarrow \eta^*_{ijk} = \text{above}$$

Gaussian Naive Bayes

- This is just a Gaussian Bayes Classifier with a separate diagonal covariance matrix for each class.
- Equivalent to fitting a one-dimensional Gaussian to each input for each possible class.
- Decision surfaces are quadratics, not linear...



Logistic/Softmax Regression

- Model: y is a multinomial random variable whose posterior is the softmax of linear functions of *any* feature vector.

$$p(y = k|\mathbf{x}, \theta) = \frac{e^{\theta_k^\top \mathbf{x}}}{\sum_j e^{\theta_j^\top \mathbf{x}}}$$

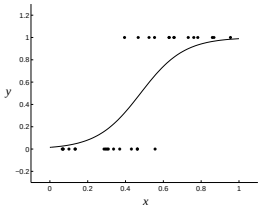
- Fitting: now we optimize the *conditional* likelihood:

$$\ell(\theta; \mathcal{D}) = \sum_{mk} [y^m = k] \log p(y = k|\mathbf{x}^m, \theta) = \sum_{mk} y_k^m \log p_k^m$$

$$\frac{\partial \ell}{\partial \theta_i} = \sum_{mk} \frac{\partial \ell_k^m}{\partial p_k^m} \frac{\partial p_k^m}{\partial z_i^m} \frac{\partial z_i^m}{\partial \theta_i}$$

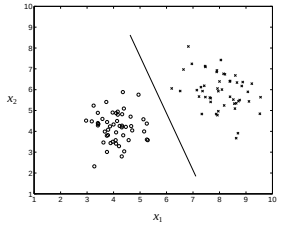
$$= \sum_{mk} \frac{y_k^m}{p_k^m} p_k^m (\delta_{ik} - p_i^m) \mathbf{x}^m$$

$$= \sum_m (y_k^m - p_k^m) \mathbf{x}^m$$



Discriminative Models

- Parametrize $p(y|\mathbf{x})$ directly, forget $p(\mathbf{x}, y)$ and Bayes' rule.
- As long as $p(y|\mathbf{x})$ or discriminants $f(y|\mathbf{x})$ are linear functions of $\mathbf{x}$ (or monotone transforms), decision surfaces will be piecewise linear.
- Don't need to model the density of the features.
  Some density models have lots of parameters.
  Many densities give same linear classifier.
  But we cannot generate new labeled data.
- Optimize a cost function closer to the one we use at test time.



More on Logistic Regression

- Hardest Part: picking the feature vector $\mathbf{x}$.
- Amazing fact: the conditional likelihood is (almost) convex in the parameters θ. Still no local minima!
- Gradient is easy to compute; so easy (if slow) to optimize using gradient descent or Newton-Raphson / IRLS.
- Why almost? Consider what happens if there are two features with identical classification patterns in our training data. Logistic Regression can only see the sum of the corresponding weights.
- Solution? Weight decay: add $\epsilon \sum \theta^2$ to the cost function, which subtracts $2\epsilon\theta$ from each gradient.
- Why is this method called logistic regression?
- It should really be called "softmax linear regression".
- Log odds (logit) between any two classes is linear in parameters.

Noisy-OR Classifier

- Many probabilistic models can be obtained as noisy versions of formulas from propositional logic.
- Noisy-OR: each input x_i activates output y w/some probability.

$$p(y=0|\mathbf{x},\alpha) = \prod_i \alpha_i^{x_i} = \exp\left\{\sum_i x_i \log \alpha_i\right\}$$

- Letting $\theta_i = -\log \alpha_i$ we get yet another linear classifier:

$$p(y=1|\mathbf{x},\theta) = 1 - e^{-\theta^\top \mathbf{x}}$$

Learning Markov Models

- The ML parameter estimates for a simple Markov model are easy:

$$p(\mathbf{y}_1,\mathbf{y}_2,\ldots,\mathbf{y}_T) = p(\mathbf{y}_1\ldots\mathbf{y}_k)\prod_{t=k+1}^{T} p(\mathbf{y}_t|\mathbf{y}_{t-1},\mathbf{y}_{t-2},\ldots,\mathbf{y}_{t-k})$$

$$\log p(\{\mathbf{y}\}) = \log p(\mathbf{y}_1\ldots\mathbf{y}_k) + \sum_{t=k+1}^{T} \log p(\mathbf{y}_t|\mathbf{y}_{t-1},\mathbf{y}_{t-2},\ldots,\mathbf{y}_{t-k})$$

- Each window of $k+1$ outputs is a training case for the model $p(\mathbf{y}_t|\mathbf{y}_{t-1},\mathbf{y}_{t-2},\ldots,\mathbf{y}_{t-k})$.
- Example: for discrete outputs (symbols) and a 2nd-order markov model we can use the multinomial model:

$$p(y_t = m|y_{t-1}=a, y_{t-2}=b) = \alpha_{mab}$$

The maximum likelihood values for α are:

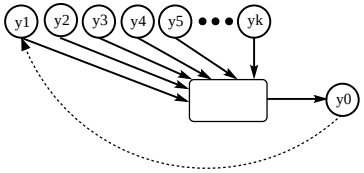
$$\alpha^*_{mab} = \frac{\text{num}[t\ s.t.\ y_t = m, y_{t-1}=a, y_{t-2}=b]}{\text{num}[t\ s.t.\ y_{t-1}=a, y_{t-2}=b]}$$

Chains: Markov Models

- If variables have some temporal/spatial order, we can model their joint distribution as a dynamical/diffusion system.
- Simple idea: next output depends only on k previous outputs:

$$\mathbf{y}_t = f[\mathbf{y}_{t-1},\mathbf{y}_{t-2},\ldots,\mathbf{y}_{t-k}]$$

k is called the *order* of the Markov Model

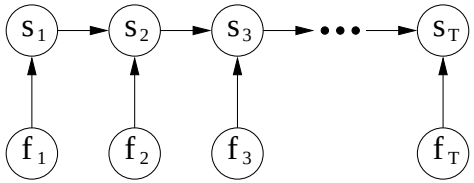


- Add noise to make the system probabilistic:

$$p(\mathbf{y}_t|\mathbf{y}_{t-1},\mathbf{y}_{t-2},\ldots,\mathbf{y}_{t-k})$$

Maximum Entropy Markov Models

- We can extend this idea to a "logistic regression through time" type of conditional model called a *maximum entropy markov model*.

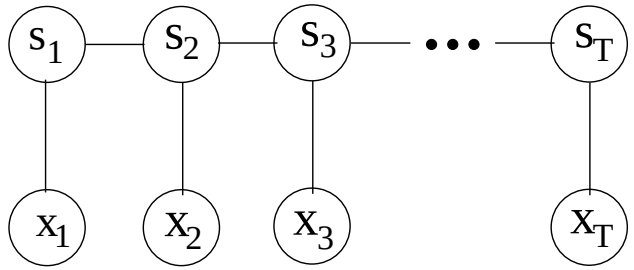


- The joint distribution is now a conditional model:

$$p(s_1^T|x_1^T) = \prod_t p(s_t|s_{t-1}, f_t(x_1^T))$$

- The features f_t can be very nonlocal functions of the underlying input sequence, for example they can consult things in the past and in the future.

Conditional Random Fields



How is this different than the MEMM?

Normalization is global and not local.

What are the cliques in this model?

Undirected Tree Graphical Models

- Undirected trees are connected, acyclic graphs with exactly (D-1) edges if there are D nodes (variables).
- For undirected trees, the cliques are all pairs of connected nodes.

$$p(\mathbf{x}) = \frac{1}{Z} \prod_i \psi_i(x_i, \mathbf{x}_{\pi_i})$$

  where we can make $Z = 1$ with the choice $\psi_i = p(x_i|\mathbf{x}_{\pi_i})$ except for one clique involving the root: $\psi_j = p(x_r)p(x_j|\mathbf{x}_{\pi_j})$
- Trees have no "explaining-away" (converging arrows). Therefore, d-separation and regular separation are equivalent.
- Directed and undirected trees are equivalent and the choice of root is arbitrary (for fully observed models).
- Another characterization of trees: there is exactly one path between any pair of nodes (without doubling back).

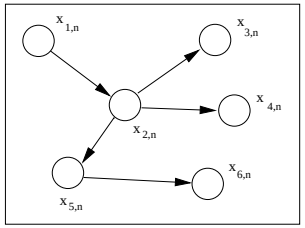
Directed Tree Graphical Models

- Directed trees are DAGMs in which each variable x_i has exactly one other variable as its parent $\mathbf{x}_{\pi_i}$ except the "root" x_{root} which has no parents. Thus, the probability of a variable taking on a certain value depends only on the value of its parent:

$$p(\mathbf{x}) = p(x_{\text{root}}) \prod_{i \neq \text{root}} p(x_i|\mathbf{x}_{\pi_i})$$

- Trees are the next step up from assuming independence. Instead of considering variables in isolation, consider them in pairs.

NB: each node (except root) has exactly one parent, but nodes may have more than one child.



Likelihood function

- Notation:
  $\mathbf{y}_i \equiv$ a node x_i and its single parent $\mathbf{x}_{\pi_i}$.
  $\mathbf{V}_i \equiv$ set of joint configurations of node i and its parent $\mathbf{x}_{\pi_i}$
  ($\mathbf{y}_{\text{root}} \equiv x_{\text{root}}$ and $\mathbf{V}_{\text{root}} \equiv \mathbf{v}_{\text{root}}$)
- Directed model likelihood:

$$\begin{aligned}
\ell(\theta; \mathcal{D}) &= \sum_n \log p(\mathbf{x}^n) = \sum_n \left[\log p_r(x_r^n) + \sum_{i \neq r} \log p(x_i{}^n|\mathbf{x}_{\pi_i}{}^n) \right] \\
&= \sum_n \sum_i \sum_{\mathbf{v} \in \mathbf{V}_i} [\mathbf{y}_i^n = \mathbf{v}] \log p_i(\mathbf{v}) \qquad \text{indicator trick} \\
&= \sum_i \sum_{\mathbf{v} \in \mathbf{V}_i} N_i(\mathbf{v}) \log p_i(\mathbf{v})
\end{aligned}$$

  where $N_i(\mathbf{v}) = \sum_n [\mathbf{y}_i^n = \mathbf{v}]$ and $p_i(\mathbf{v}_i) = p(x_i|\mathbf{x}_{\pi_i})$.
- Trees are in the exponential family with $\mathbf{y}_i$ as sufficient statistics.

Maximum Likelihood Parameters Given Structure

- Trees are just a special case of fully observed graphical models.
- For discrete data x_i with values v_i, each node stores a conditional probability table (CPT) over its values given its parent's value. The ML parameter estimates are just the empirical histograms of each node's values given its parent:

$$p^*(x_i = v_i | \mathbf{x}_{\pi_i} = v_j) = \frac{N(x_i = v_i, \mathbf{x}_{\pi_i} = v_j)}{\sum_{\mathbf{v}_i} N(x_i = v_i, \mathbf{x}_{\pi_i} = v_j)} = \frac{N_i(\mathbf{y}_i)}{N_{\pi_i}(v_j)}$$

  except for the root which uses marginal counts $N_r(v_r)/N$.
- For continuous data, the most common model is a two-dimensional Gaussian at each node. The ML parameters are just to set the mean of $p_i(\mathbf{y}_i)$ to be the sample mean of $[x_i; \mathbf{x}_{\pi_i}]$ and the covariance matrix to the sample covariance.
- In practice we should use some kind of smoothing/regularization.

Optimal Structure

- Let us rewrite the likelihood function:

$$\begin{aligned}\ell(\theta; \mathcal{D}) &= \sum_{\mathbf{x} \in \mathbf{V}_{\text{all}}} N(\mathbf{x}) \log p(\mathbf{x}) \\ &= \sum_{\mathbf{x}} N(\mathbf{x}) \left(\log p(\mathbf{x}_r) + \sum_{i \neq r} \log p(x_i | \mathbf{x}_{\pi_i}) \right)\end{aligned}$$

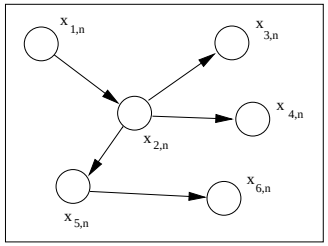
- ML parameters, are equal to the observed frequency counts $q(\cdot)$:

$$\begin{aligned}\frac{\ell^*}{N} &= \sum_{\mathbf{x} \in \mathbf{V}_{\text{all}}} q(\mathbf{x}) \left(\log q(\mathbf{x}_r) + \sum_{i \neq r} \log q(x_i | \mathbf{x}_{\pi_i}) \right) \\ &= \sum_{\mathbf{x}} q(\mathbf{x}) \left(\log q(\mathbf{x}_r) + \sum_{i \neq r} \log \frac{q(x_i, \mathbf{x}_{\pi_i})}{q(\mathbf{x}_{\pi_i})} \right) \\ &= \sum_{\mathbf{x}} q(\mathbf{x}) \sum_{i \neq r} \log \frac{q(x_i, \mathbf{x}_{\pi_i})}{q(x_i) q(\mathbf{x}_{\pi_i})} + \sum_{\mathbf{x}} q(\mathbf{x}) \sum_i \log q(\mathbf{x}_i)\end{aligned}$$

- NB: second term does not depend on structure.

Structure Learning

- What about the tree structure (links)?
  How do we know which nodes to make parents of which?



- Bold idea: how can we also *learn* the optimal structure?
  In principle, we could search all combinatorial structures, for each compute the ML parameters, and take the best one.
- But is there a better way? Yes. It turns out that structure learning in tree models can be converted to a good old computer science problem: maximum weight spanning tree.

Edge Weights

- Each term in sum $i \neq r$ corresponds to an edge from i to its parent.

$$\begin{aligned}\frac{\ell^*}{N} &= \sum_{\mathbf{x}} q(\mathbf{x}) \sum_{i \neq r} \log \frac{q(x_i, \mathbf{x}_{\pi_i})}{q(x_i) q(\mathbf{x}_{\pi_i})} + C \\ &= \sum_{i \neq r} \sum_{x_i, \mathbf{x}_{\pi_i}} q(x_i, \mathbf{x}_{\pi_i}) \log \frac{q(x_i, \mathbf{x}_{\pi_i})}{q(x_i) q(\mathbf{x}_{\pi_i})} + C \\ &= \sum_{i \neq r} \sum_{\mathbf{y}_i} q(\mathbf{y}_i) \log \frac{q(\mathbf{y}_i)}{q(x_i) q(\mathbf{x}_{\pi_i})} + C \\ &= \sum_{i \neq r} W(i; \pi_i) + C\end{aligned}$$

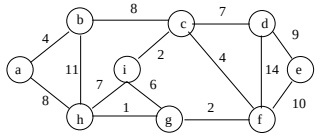
  where the edge weights W are defined by *mutual information*:

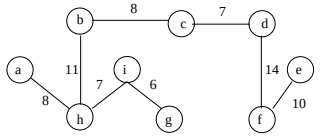
$$W(i; j) = \sum_{x_i, x_j} q(x_i, x_j) \log \frac{q(x_i, x_j)}{q(x_i) q(x_j)}$$

- So overall likelihood is sum of weights on edges that we use.
  We need the maximum weight spanning tree.

Kruskal's algorithm (Greedy Search)

- To find the maximum weight spanning tree A on a graph with nodes U and weighted edges E:
  1. $A \leftarrow$ empty
  2. Sort edges E by nonincreasing weight: $e_1, e_2, \ldots, e_K$.
  3. for $k = 1$ to K $\{A \mathrel{+}= e_k$ unless doing so creates a cycle$\}$





Maximum Likelihood Trees

We can now completely solve the tree learning problem:

1. Compute the marginal counts $q(x_i)$ for each node and pairwise counts $q(x_i, x_j)$ for all pairs of nodes.
2. Set the weights to the mutual informations:

$$W(i;j) = \sum_{x_i, x_j} q(x_i, x_j) \log \frac{q(x_i, x_j)}{q(x_i)q(x_j)}$$

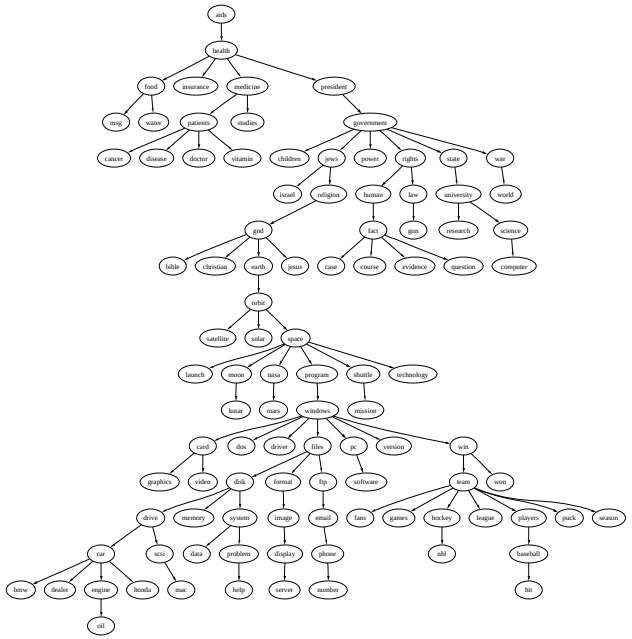
3. Find the maximum weight spanning tree A=MWST(W).
4. Using the undirected tree A chosen by MWST, pick a root arbitrarily and orient the edges away from the root. Set the conditional functions to the observed frequencies:

$$p(x_i|\mathbf{x}_{\pi_i}) = \frac{q(x_i, \mathbf{x}_{\pi_i})}{\sum_{x_i} q(x_i, \mathbf{x}_{\pi_i})} = \frac{q(x_i, \mathbf{x}_{\pi_i})}{q(\mathbf{x}_{\pi_i})}$$

Notes

- *Any* directed tree consistent with the undirected tree found by the algorithm above will assign the same likelihood to any dataset.
- Amazingly, as far as likelihood goes, the root is arbitrary. We can just pick one node and orient the edges away from it. Or we can work with undirected models.
- For continuous nodes (e.g. Gaussian), the situation is similar, except that computing the mutual information requires an integral.
- Mutual information is the *Kullback-Leibler* divergence (cross-entropy) between a distribution and the product of its marginals. Measures how far from independent the joint distribution is.

$$W(i;j) = \mathrm{I}[x_i; x_j] = \mathrm{KL}[q(x_i, xj) \| q(x_i)q(x_j)]$$



Lecture 3:

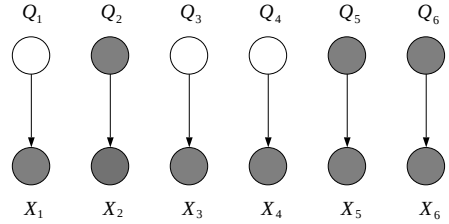
Latent Variables Models and Learning with the EM Algorithm

Sam Roweis

Thursday January 27, 2005
Machine Learning Summer School

Unobserved Variables

- We have been assuming that we observe all the random variables in our model at training time, and all the "inputs" at test time.
- But certain variables Q in our models may be *unobserved*, either some of the time or always, either at training time or at test time.



(Graphically, we will use shading to indicate observation.)

Fundamental Operations

- What can we do with a probabilistic graphical model?
- *Generate new data.*
  For this you need to know how to sample from local models (directed) or how to do Gibbs or other sampling (undirected).
- *Novelty/outlier detection.*
  When all nodes are either observed or marginalized the result is a single number which is the log prob of the configuration.
- *Parameter Learning from Examples.*
  Set the parameters of the local functions given some (partially) observed data to maximize the probability of seeing that data.
- *Inference of Internal/Hidden Variables.*
  Compute expectations of some nodes given others which are observed or marginalized.

Partially Unobserved (Missing) Variables

- If variables are occasionally unobserved they are *missing data*.
  e.g. undefinied inputs, missing class labels, erroneous target values
- In this case, we can still model the joint distribution, but we define a new cost function in which we *sum out* or *marginalize* the missing values at training or test time:

$$\ell(\theta;\mathcal{D}) = \sum_{\text{complete}} \log p(\mathbf{x}^c,\mathbf{y}^c|\theta) + \sum_{\text{missing}} \log p(\mathbf{x}^m|\theta)$$
$$= \sum_{\text{complete}} \log p(\mathbf{x}^c,\mathbf{y}^c|\theta) + \sum_{\text{missing}} \log \sum_{\mathbf{y}} p(\mathbf{x}^m,\mathbf{y}|\theta)$$

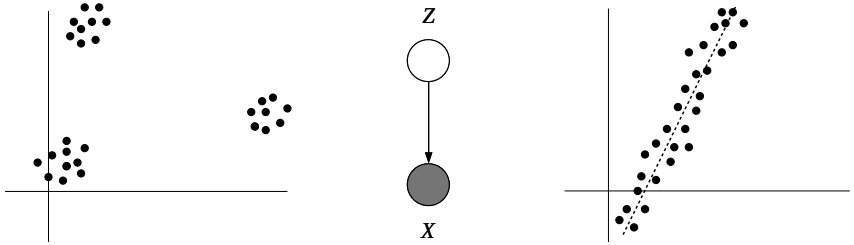
[Recall that $p(x) = \sum_q p(x,q)$.]

Latent Variables

- What to do when a variable $\mathbf{z}$ is *always* unobserved?
  Depends on where it appears in our model. If we never condition on it when computing the probability of the variables we *do* observe, then we can just forget about it and integrate it out.
  e.g. given $\mathbf{y}, \mathbf{x}$ fit the model $p(\mathbf{z}, \mathbf{y}|\mathbf{x}) = p(\mathbf{z}|\mathbf{y})p(\mathbf{y}|\mathbf{x}, \mathbf{w})p(\mathbf{w})$.
  (In other words if it is a leaf node.)
- But if $\mathbf{z}$ is conditioned on, we need to model it:
  e.g. given $\mathbf{y}, \mathbf{x}$ fit the model $p(\mathbf{y}|\mathbf{x}) = \sum_{\mathbf{z}} p(\mathbf{y}|\mathbf{x}, \mathbf{z})p(\mathbf{z})$

Clustering vs. Classification
Latent Factor Models vs. Regression

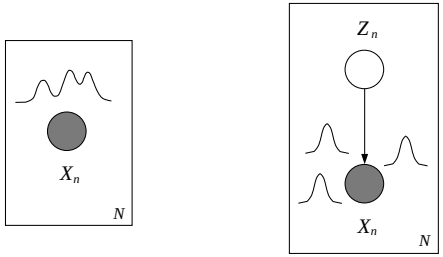
- You can think of clustering as the problem of classification with missing class labels.



- You can think of factor models (such as factor analysis, PCA, ICA, etc.) as linear or nonlinear regression with missing inputs.

Where Do Latent Variables Come From?

- Latent variables may appear naturally, from the structure of the problem, because something wasn't measured, because of faulty sensors, occlusion, privacy, etc.
- But also, we may want to *intentionally* introduce latent variables to model complex dependencies between variables without looking at the dependencies between them directly.
  This can actually simplify the model (e.g. mixtures).



(a) (b)

Why is Learning Harder?

- In fully observed iid settings, the probability model is a product thus the log likelihood is a sum where terms decouple.
  (At least for directed models.)

$$\begin{aligned}\ell(\theta; \mathcal{D}) &= \log p(\mathbf{x}, \mathbf{z}|\theta) \\ &= \log p(\mathbf{z}|\theta_z) + \log p(\mathbf{x}|\mathbf{z}, \theta_x)\end{aligned}$$

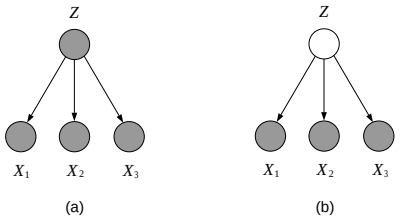
- With latent variables, the probability already contains a sum, so the log likelihood has all parameters coupled together via $\log \sum()$:

$$\begin{aligned}\ell(\theta; \mathcal{D}) &= \log \sum_{\mathbf{z}} p(\mathbf{x}, \mathbf{z}|\theta) \\ &= \log \sum_{\mathbf{z}} p(\mathbf{z}|\theta_z)p(\mathbf{x}|\mathbf{z}, \theta_x)\end{aligned}$$

  (Just as with the partition function in undirected models.)

Learning with Latent Variables

- Likelihood $\ell(\theta; \mathcal{D}) = \log \sum_{\mathbf{z}} p(\mathbf{z}|\theta_z) p(\mathbf{x}|\mathbf{z}, \theta_x)$ couples parameters:



- We can treat this as a black box probability function and just try to optimize the likelihood as a function of θ (e.g. gradient descent). However, sometimes taking advantage of the latent variable structure can make parameter estimation easier.
- Good news: soon we will see the *EM algorithm* which allows us to treat learning with latent variables using fully observed tools.
- Basic trick: guess the values you don't know.
  Basic math: use convexity to lower bound the likelihood.

Mixture Densities

- Exactly like a classification model but the class is unobserved and so we sum it out. What we get is a perfectly valid density:

$$\begin{aligned} p(\mathbf{x}|\theta) &= \sum_{k=1}^{K} p(z=k|\theta_z) p(\mathbf{x}|z=k, \theta_k) \\ &= \sum_{k} \alpha_k p_k(\mathbf{x}|\theta_k) \end{aligned}$$

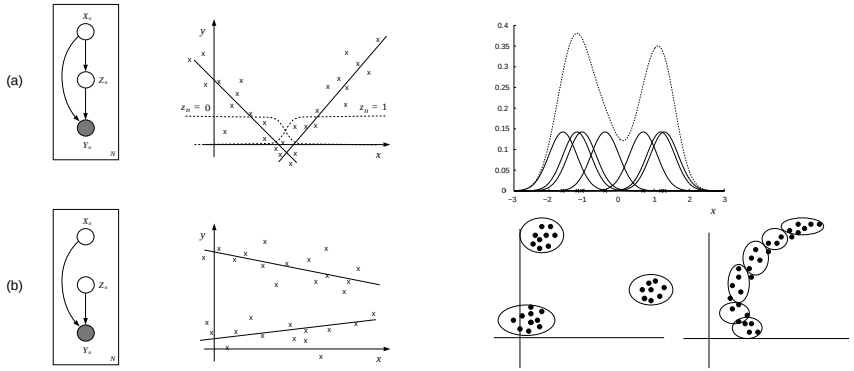
  where the "mixing proportions" add to one: $\sum_k \alpha_k = 1$.
- We can use Bayes' rule to compute the posterior probability of the mixture component given some data:

$$p(z=k|\mathbf{x}, \theta) = \frac{\alpha_k p_k(\mathbf{x}|\theta_k)}{\sum_j \alpha_j p_j(\mathbf{x}|\theta_j)}$$

  these quantities are called *responsibilities*.

Mixture Models (1 discrete latent var)

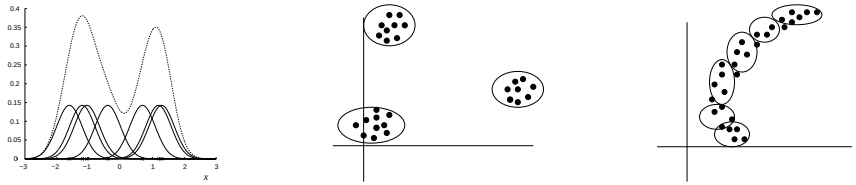
- Most basic latent variable model with a single discrete node z.
- Allows different submodels (experts) to contribute to the (conditional) density model in different parts of the space.
- Divide and conquer idea: use simple parts to build complex models. (e.g. multimodal densities, or piecewise-linear regressions).



Clustering Example: Gaussian Mixture Models

- Consider a mixture of K Gaussian components:

$$\begin{aligned} p(\mathbf{x}|\theta) &= \sum_{k} \alpha_k \mathcal{N}(\mathbf{x}|\mu_k, \Sigma_k) \\ p(z=k|\mathbf{x}, \theta) &= \frac{\alpha_k \mathcal{N}(\mathbf{x}|\mu_k, \Sigma_k)}{\sum_j \alpha_j \mathcal{N}(\mathbf{x}|\mu_k, \Sigma_k)} \\ \ell(\theta; \mathcal{D}) &= \sum_{n} \log \sum_{k} \alpha_k \mathcal{N}(\mathbf{x}^n|\mu_k, \Sigma_k) \end{aligned}$$



- Density model: $p(x|\theta)$ is a familiarity signal.
  Clustering: $p(z|\mathbf{x}, \theta)$ is the assignment rule, $-\ell(\theta)$ is the cost.

Regression Example: Mixtures of Experts

- Also called conditional mixtures. Exactly like a class-conditional model but the class is unobserved and so we sum it out again:

$$p(\mathbf{y}|\mathbf{x},\theta) = \sum_{k=1}^{K} p(z=k|\mathbf{x},\theta_z)p(\mathbf{y}|z=k,\mathbf{x},\theta_k)$$
$$= \sum_k \alpha_k(\mathbf{x}|\theta_z)p_k(\mathbf{y}|\mathbf{x},\theta_k)$$

  where $\sum_k \alpha_k(\mathbf{x}) = 1 \quad \forall \mathbf{x}$.
- Harder: must learn $\alpha(\mathbf{x})$ (unless chose z independent of $\mathbf{x}$).
- We can still use Bayes' rule to compute the posterior probability of the mixture component given some data:

$$p(z=k|\mathbf{x},\mathbf{y},\theta) = \frac{\alpha_k(\mathbf{x})p_k(\mathbf{y}|\mathbf{x},\theta_k)}{\sum_j \alpha_j(\mathbf{x})p_j(\mathbf{y}|\mathbf{x},\theta_j)}$$

  this function is often called the *gating function*.

Gradient Learning with Mixtures

- We can learn mixture densities using gradient descent on the likelihood as usual. The gradients are quite interesting:

$$\ell(\theta) = \log p(\mathbf{x}|\theta) = \log \sum_k \alpha_k p_k(\mathbf{x}|\theta_k)$$
$$\frac{\partial \ell}{\partial \theta} = \frac{1}{p(\mathbf{x}|\theta)} \sum_k \alpha_k \frac{\partial p_k(\mathbf{x}|\theta_k)}{\partial \theta}$$
$$= \sum_k \alpha_k \frac{1}{p(\mathbf{x}|\theta)} p_k(\mathbf{x}|\theta_k) \frac{\partial \log p_k(\mathbf{x}|\theta_k)}{\partial \theta}$$
$$= \sum_k \alpha_k \frac{p_k(\mathbf{x}|\theta_k)}{p(\mathbf{x}|\theta)} \frac{\partial \ell_k}{\partial \theta_k} = \sum_k \alpha_k r_k \frac{\partial \ell_k}{\partial \theta_k}$$

- In other words, the gradient is the *responsibility weighted sum* of the individual log likelihood gradients.

Example: Mixture of Linear Regression Experts

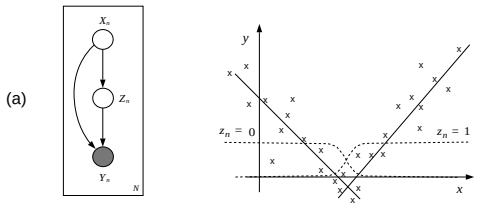
- Each expert generates data according to a linear function of the input plus additive Gaussian noise:

$$p(y|\mathbf{x},\theta) = \sum_k \alpha_k(\mathbf{x})\mathcal{N}(y|\beta_k^\top \mathbf{x}, \sigma_k^2)$$

- The "gate" function can be a softmax classification machine:

$$\alpha_k(\mathbf{x}) = p(z=k|\mathbf{x}) = \frac{e^{\eta_k^\top \mathbf{x}}}{\sum_j e^{\eta_j^\top \mathbf{x}}}$$

- Remember: we are not modeling the density of the inputs $\mathbf{x}$.



Reminder: Learning with Latent Variables

- With latent variables, the probability contains a sum, so the log likelihood has all parameters coupled together:

$$\ell(\theta;\mathcal{D}) = \log \sum_{\mathbf{z}} p(\mathbf{x},\mathbf{z}|\theta) = \log \sum_{\mathbf{z}} p(\mathbf{z}|\theta_z)p(\mathbf{x}|\mathbf{z},\theta_x)$$

  (we can also consider continuous $\mathbf{z}$ and replace $\sum$ with $\int$)
- If the latent variables were observed, parameters would decouple again and learning would be easy:

$$\ell(\theta;\mathcal{D}) = \log p(\mathbf{x},\mathbf{z}|\theta) = \log p(\mathbf{z}|\theta_z) + \log p(\mathbf{x}|\mathbf{z},\theta_x)$$

- One idea: ignore this fact, compute $\partial\ell/\partial\theta$, and do learning with a smart optimizer like conjugate gradient.
- Another idea: what if we use our current parameters to *guess* the values of the latent variables, and then do fully-observed learning? This back-and-forth trick might make optimization easier.

Expectation-Maximization (EM) Algorithm

- Iterative algorithm with two linked steps:
  E-step: fill in values of $\hat{\mathbf{z}}^t$ using $p(\mathbf{z}|\mathbf{x}, \theta^t)$.
  M-step: update parameters using $\theta^{t+1} \leftarrow \text{argmax}\ \ell(\theta; \mathbf{x}, \hat{\mathbf{z}}^t)$.
- E-step involves inference, which we need to do at runtime anyway. M-step is no harder than in fully observed case.
- We will prove that this procedure monotonically improves ℓ (or leaves it unchanged). Thus it always converges to a local optimum of the likelihood (as any optimizer should).
- Note: EM is an optimization strategy for objective functions that can be interpreted as likelihoods in the presence of missing data.
- EM is *not* a cost function such as "maximum-likelihood". EM is *not* a model such as "mixture-of-Gaussians".

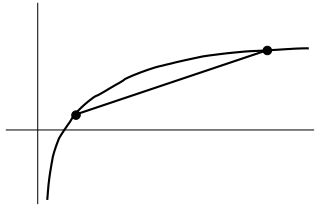
Expected Complete Log Likelihood

- For *any* distribution $q(\mathbf{z})$ define *expected complete log likelihood*:

$$\ell_q(\theta; \mathbf{x}) = \langle \ell_c(\theta; \mathbf{x}, \mathbf{z}) \rangle_q \equiv \sum_{\mathbf{z}} q(\mathbf{z}|\mathbf{x}) \log p(\mathbf{x}, \mathbf{z}|\theta)$$

- Amazing fact: $\ell(\theta) \geq \ell_q(\theta) + \mathcal{H}(q)$ because of concavity of $\log$:

$$\begin{aligned}
\ell(\theta; \mathbf{x}) &= \log p(\mathbf{x}|\theta) \\
&= \log \sum_{\mathbf{z}} p(\mathbf{x}, \mathbf{z}|\theta) \\
&= \log \sum_{\mathbf{z}} q(\mathbf{z}|\mathbf{x}) \frac{p(\mathbf{x}, \mathbf{z}|\theta)}{q(\mathbf{z}|\mathbf{x})} \\
&\geq \sum_{\mathbf{z}} q(\mathbf{z}|\mathbf{x}) \log \frac{p(\mathbf{x}, \mathbf{z}|\theta)}{q(\mathbf{z}|\mathbf{x})}
\end{aligned}$$



- Where the inequality is called *Jensen's inequality*. (It is only true for distributions: $\sum q(\mathbf{z}) = 1;\ q(\mathbf{z}) > 0$.)

Complete & Incomplete Log Likelihoods

- Observed variables $\mathbf{x}$, latent variables $\mathbf{z}$, parameters θ:

$$\ell_c(\theta; \mathbf{x}, \mathbf{z}) = \log p(\mathbf{x}, \mathbf{z}|\theta)$$

  is the *complete log likelihood*.
- Usually optimizing $\ell_c(\theta)$ given both $\mathbf{z}$ and $\mathbf{x}$ is straightforward. (e.g. class conditional Gaussian fitting, linear regression)
- With $\mathbf{z}$ unobserved, we need the log of a marginal probability:

$$\ell(\theta; \mathbf{x}) = \log p(\mathbf{x}|\theta) = \log \sum_{\mathbf{z}} p(\mathbf{x}, \mathbf{z}|\theta)$$

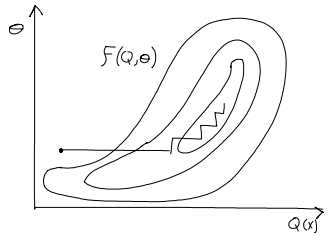
  which is the *incomplete log likelihood*.

Lower Bounds and Free Energy

- For fixed data $\mathbf{x}$, define a functional called the *free energy*:

$$F(q, \theta) \equiv \sum_{\mathbf{z}} q(\mathbf{z}|\mathbf{x}) \log \frac{p(\mathbf{x}, \mathbf{z}|\theta)}{q(\mathbf{z}|\mathbf{x})} \leq \ell(\theta)$$

- The EM algorithm is coordinate-ascent on F:
  E-step: $q^{t+1} = \text{argmax}_q \ F(q, \theta^t)$
  M-step: $\theta^{t+1} = \text{argmax}_\theta \ F(q^{t+1}, \theta^t)$



M-step: maximization of expected ℓ_c

- Note that the free energy breaks into two terms:

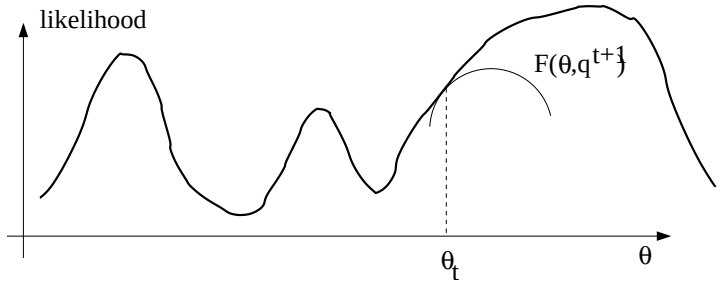
$$\begin{aligned} F(q,\theta) &= \sum_{\mathbf{z}} q(\mathbf{z}|\mathbf{x}) \log \frac{p(\mathbf{x},\mathbf{z}|\theta)}{q(\mathbf{z}|\mathbf{x})} \\ &= \sum_{\mathbf{z}} q(\mathbf{z}|\mathbf{x}) \log p(\mathbf{x},\mathbf{z}|\theta) - \sum_{\mathbf{z}} q(\mathbf{z}|\mathbf{x}) \log q(\mathbf{z}|\mathbf{x}) \\ &= \ell_q(\theta;\mathbf{x}) + \mathcal{H}(q) \end{aligned}$$

  (this is where its name comes from)
- The first term is the expected complete log likelihood (energy) and the second term, which does not depend on θ, is the entropy.
- Thus, in the M-step, maximizing with respect to θ for fixed q we only need to consider the first term:

$$\theta^{t+1} = \text{argmax}_\theta \, \ell_q(\theta;\mathbf{x}) = \text{argmax}_\theta \sum_{\mathbf{z}} q(\mathbf{z}|\mathbf{x}) \log p(\mathbf{x},\mathbf{z}|\theta)$$

EM Constructs Sequential Convex Lower Bounds

- Consider the likelihood function and the function $F(q^{t+1},\cdot)$.



E-step: inferring latent posterior

- Claim: the optimim setting of q in the E-step is:

$$q^{t+1} = p(\mathbf{z}|\mathbf{x},\theta^t)$$

- This is the posterior distribution over the latent variables given the data and the parameters. Often we need this at test time anyway (e.g. to perform classification).
- Proof (easy): this setting saturates the bound $\ell(\theta;\mathbf{x}) \geq F(q,\theta)$

$$\begin{aligned} F(p(\mathbf{z}|\mathbf{x},\theta^t),\theta^t) &= \sum_{\mathbf{z}} p(\mathbf{z}|\mathbf{x},\theta^t) \log \frac{p(\mathbf{x},\mathbf{z}|\theta^t)}{p(\mathbf{z}|\mathbf{x},\theta^t)} \\ &= \sum_{\mathbf{z}} p(\mathbf{z}|\mathbf{x},\theta^t) \log p(\mathbf{x}|\theta^t) \\ &= \log p(\mathbf{x}|\theta^t) \textstyle\sum_{\mathbf{z}} p(\mathbf{z}|\mathbf{x},\theta^t) \\ &= \ell(\theta;\mathbf{x}) \cdot 1 \end{aligned}$$

- Can also show this result using variational calculus or the fact that $\ell(\theta) - F(q,\theta) = \text{KL}[q||p(\mathbf{z}|\mathbf{x},\theta)]$

Example: Mixtures of Gaussians

- Recall: a mixture of K Gaussians:
  $p(\mathbf{x}|\theta) = \sum_k \alpha_k \mathcal{N}(\mathbf{x}|\mu_k,\Sigma_k)$
  $\ell(\theta;\mathcal{D}) = \sum_n \log \sum_k \alpha_k \mathcal{N}(\mathbf{x}^n|\mu_k,\Sigma_k)$
- Learning with EM algorithm:

$$\mathbf{E-step}: \quad p_{kn}^t = \mathcal{N}(\mathbf{x}^n|\mu_k^t,\Sigma_k^t)$$

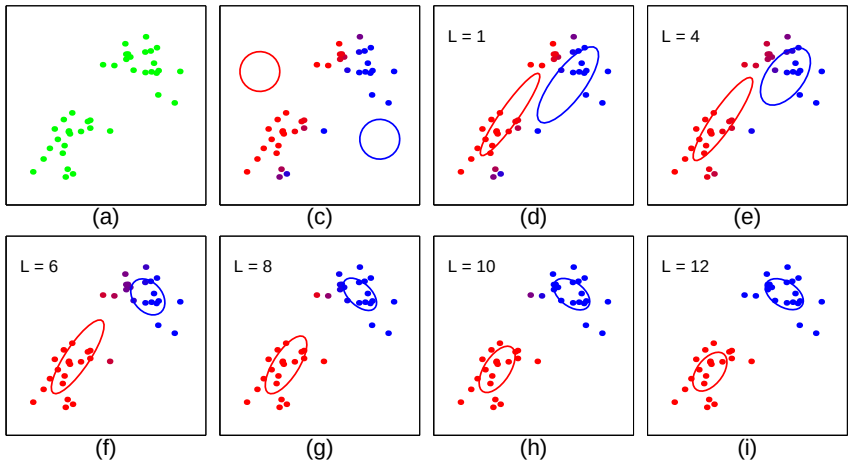
$$q_{kn}^{t+1} = p(z{=}k|\mathbf{x}^n,\theta^t) = \frac{\alpha_k^t p_{kn}^t}{\sum_j \alpha_j^t p_{kn}^t}$$

$$\mathbf{M-step}: \quad \mu_k^{t+1} = \frac{\sum_n q_{kn}^{t+1} \mathbf{x}^n}{\sum_n q_{kn}^{t+1}}$$

$$\Sigma_k^{t+1} = \frac{\sum_n q_{kn}^{t+1} (\mathbf{x}^n - \mu_k^{t+1})(\mathbf{x}^n - \mu_k^{t+1})^\top}{\sum_n q_{kn}^{t+1}}$$

$$\alpha_k^{t+1} = \frac{1}{M} \sum_n q_{kn}^{t+1}$$

EM for MOG



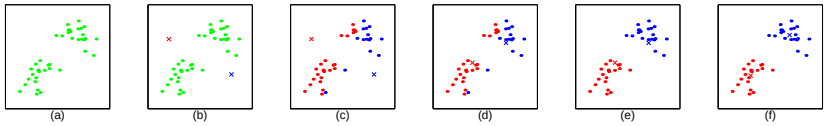
Compare: K-means

- The EM algorithm for mixtures of Gaussians is just like a soft version of the K-means algorithm.
- In the K-means "E-step" we do hard assignment:

$$c_n^{t+1} = \text{argmin}_k (\mathbf{x}^n - \mu_k^t)^\top \Sigma_k^{-1} (\mathbf{x}^n - \mu_k^t)$$

- In the K-means "M-step" we update the means as the weighted sum of the data, but now the weights are 0 or 1:

$$\mu_k^{t+1} = \frac{\sum_n [c_k^{t+1} = n] \mathbf{x}^n}{\sum_n [c_k^{t+1} = n]}$$



Derivation of M-step

- Expected complete log likelihood $\ell_q(\theta; \mathcal{D})$:

$$\sum_n \sum_k q_{kn} \left[\log \alpha_k - \frac{1}{2} (\mathbf{x}^n - \mu_k^{t+1})^\top \Sigma_k^{-1} (\mathbf{x}^n - \mu_k^{t+1}) - \frac{1}{2} \log |2\pi\Sigma_k| \right]$$

- For fixed q we can optimize the parameters:

$$\frac{\partial \ell_q}{\partial \mu_k} = \Sigma_k^{-1} \sum_n q_{kn} (\mathbf{x}^n - \mu_k)$$

$$\frac{\partial \ell_q}{\partial \Sigma_k^{-1}} = \frac{1}{2} \sum_n q_{kn} \left[\Sigma_k^\top - (\mathbf{x}^n - \mu_k^{t+1})(\mathbf{x}^n - \mu_k^{t+1})^\top \right]$$

$$\frac{\partial \ell_q}{\partial \alpha_k} = \frac{1}{\alpha_k} \sum_n q_{kn} - \lambda \qquad (\lambda = M)$$

- Fact: $\frac{\partial \log |A^{-1}|}{\partial A^{-1}} = A^\top$ and $\frac{\partial \mathbf{x}^\top A \mathbf{x}}{\partial A} = \mathbf{x}\mathbf{x}^\top$

Variants

- Sparse EM:
  Do not recompute exactly the posterior probability on each data point under all models, because it is almost zero.
  Instead keep an "active list" which you update every once in a while.
- Generalized (Incomplete) EM: It might be hard to find the ML parameters in the M-step, even given the completed data. We can still make progress by doing an M-step that improves the likelihood a bit (e.g. gradient step).

Recap: EM Algorithm

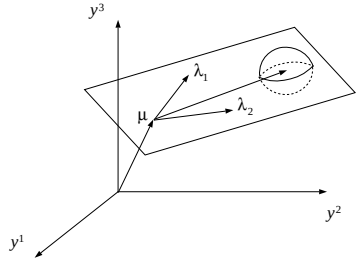
- A way of maximizing likelihood function for latent variable models. Finds ML parameters when the original (hard) problem can be broken up into two (easy) pieces:
  1. Estimate some "missing" or "unobserved" data from observed data and current parameters.
  2. Using this "complete" data, find the maximum likelihood parameter estimates.
- Alternate between filling in the latent variables using our best guess (posterior) and updating the paramters based on this guess:
  E-step: $q^{t+1} = p(\mathbf{z}|\mathbf{x}, \theta^t)$
  M-step: $\theta^{t+1} = \text{argmax}_\theta \sum_\mathbf{z} q(\mathbf{z}|\mathbf{x}) \log p(\mathbf{x}, \mathbf{z}|\theta)$
- In the M-step we optimize a lower bound on the likelihood. In the E-step we close the gap, making bound=likelihood.

Be careful: logsum

- Often you can easily compute $b_k = \log p(\mathbf{x}|z = k, \theta_k)$, but it will be very negative, say -10^6 or smaller.
- Now, to compute $\ell = \log p(\mathbf{x}|\theta)$ you need to compute $\log \sum_k e^{b_k}$.
- Careful! Do not compute this by doing `log(sum(exp(b)))`. You will get underflow and an incorrect answer.
- Instead do this:
  - Add a constant exponent B to all the values b_k such that the largest value comes close to the maxiumum exponent allowed by machine precision: `B = MAXEXPONENT-log(K)-max(b)`.
  - Compute `log(sum(exp(b+B)))-B`.
- Example: if $\log p(x|z = 1) = -420$ and $\log p(x|z = 2) = -420$, what is $\log p(x) = \log [p(x|z = 1) + p(x|z = 2)]$?
  Answer: $\log[2e^{-420}] = -420 + \log 2$.

Continuous Latent Variables

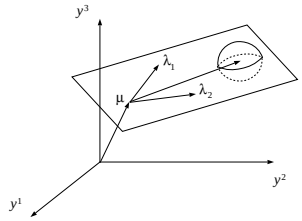
- In many models there are some *underlying causes* of the data.
- Mixture models use a discrete class variable: clustering.
- Sometimes, it is more appropriate to think in terms of continuous *factors* which control the data we observe. Geometrically, this is equivalent to thinking of a data *manifold* or subspace.



- To generate data, first generate a point within the manifold then add noise. Coordinates of point are components of latent variable.

Factor Analysis

- When we assume that the subspace is *linear* and that the underlying latent variable has a Gaussian distribution we get a model known as *factor analysis*:
  — data $\mathbf{y}$ (p-dim);
  — latent variable $\mathbf{x}$ (k-dim)



$$p(\mathbf{x}) = \mathcal{N}(\mathbf{x}|0, I)$$
$$p(\mathbf{y}|\mathbf{x}, \theta) = \mathcal{N}(\mathbf{y}|\mu + \Lambda\mathbf{x}, \Psi)$$

where μ is the mean vector, Λ is the p by k *factor loading matrix*, and Ψ is the *sensor noise covariance* (ususally diagonal).

- Important: since the product of Gaussians is still Gaussian, the joint distribution $p(\mathbf{x}, \mathbf{y})$, the other marginal $p(\mathbf{y})$ and the conditional $p(\mathbf{x}|\mathbf{y})$ are also Gaussian.

Marginal Data Distribution

- Just as with discrete latent variables, we can compute the marginal density $p(\mathbf{y}|\theta)$ by summing out $\mathbf{x}$. But now the sum is an integral:

$$p(\mathbf{y}|\theta) = \int_{\mathbf{x}} p(\mathbf{x})p(\mathbf{y}|\mathbf{x},\theta)d\mathbf{x} = \mathcal{N}(\mathbf{y}|\mu\ ,\ \Lambda\Lambda^\top + \Psi)$$

  which can be done by completing the square in the exponent.

- However, since the marginal is Gaussian, we can also just compute its mean and covariance. (Assume noise uncorrelated with data.)

$$\begin{aligned}
E[\mathbf{y}] &= E[\mu + \Lambda\mathbf{x} + \text{noise}] = \mu + \Lambda E[\mathbf{x}] + E[\text{noise}] \\
&= \mu + \Lambda \cdot 0 + 0 = \mu \\
\text{Cov}[\mathbf{y}] &= E[(\mathbf{y}-\mu)(\mathbf{y}-\mu)^\top] \\
&= E[(\mu + \Lambda\mathbf{x} + \text{noise} - \mu)(\mu + \Lambda\mathbf{x} + \text{noise} - \mu)^\top] \\
&= E[(\Lambda\mathbf{x} + n)(\Lambda\mathbf{x} + n)^\top] = \Lambda E(\mathbf{x}\mathbf{x}^\top)\Lambda^\top + E(nn^\top) \\
&= \Lambda\Lambda^\top + \Psi
\end{aligned}$$

Likelihood Function

- Since the FA data model is Gaussian, likelihood function is simple:

$$\begin{aligned}
\ell(\theta;\mathcal{D}) &= -\frac{N}{2}\log|\Lambda\Lambda^\top + \Psi| - \frac{1}{2}\sum_n (\mathbf{y}^n - \mu)^\top(\Lambda\Lambda^\top + \Psi)^{-1}(\mathbf{y}^n - \mu) \\
&= -\frac{N}{2}\log|\mathbf{V}| - \frac{1}{2}\text{trace}\left[\mathbf{V}^{-1}\sum_n (\mathbf{y}^n - \mu)(\mathbf{y}^n - \mu)^\top\right] \\
&= -\frac{N}{2}\log|\mathbf{V}| - \frac{1}{2}\text{trace}\left[\mathbf{V}^{-1}\mathbf{S}\right]
\end{aligned}$$

  $\mathbf{V}$ is model covariance; $\mathbf{S}$ is sample data covariance.

- In other words, we are trying to make the constrained model covariance as close as possible to the observed covariance, where "close" means the trace of the ratio.

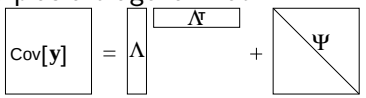
- Thus, the sufficient statistics are the same as for the Gaussian: mean $\sum_n \mathbf{y}^n$ and covariance $\sum_n (\mathbf{y}^n - \mu)(\mathbf{y}^n - \mu)^\top$.

FA = Constrained Covariance Gaussian

- Marginal density for factor analysis ($\mathbf{y}$ is p-dim, $\mathbf{x}$ is k-dim):

$$p(\mathbf{y}|\theta) = \mathcal{N}(\mathbf{y}|\mu\ ,\ \Lambda\Lambda^\top + \Psi)$$

- So the effective covariance is the low-rank outer product of two long skinny matrices plus a diagonal matrix:



- In other words, factor analysis is just a constrained Gaussian model. (If Ψ were not diagonal then we could model any Gaussian and it would be pointless.)

- Learning: how should we fit the ML parameters?

- It is easy to find μ: just take the mean of the data.
  From now on assume we have done this and re-centred $\mathbf{y}$.

- What about the other parameters?

EM for Factor Analysis

- We will do maximum likelihood learning using (surprise, surprise) the EM algorithm.
  E-step: $q_n^{t+1} = p(\mathbf{x}^n|\mathbf{y}^n,\theta^t)$
  M-step: $\theta^{t+1} = \text{argmax}_\theta \sum_n \int_{\mathbf{x}} q^{t+1}(\mathbf{x}^n|\mathbf{y}^n)\log p(\mathbf{y}^n,\mathbf{x}^n|\theta)d\mathbf{x}^n$

- For E-step we need the conditional distribution (inference)
  For M-step we need the expected log of the complete data.

$$\begin{aligned}
\mathbf{E-step}: q_n^{t+1} &= p(\mathbf{x}^n|\mathbf{y}^n,\theta^t) = \mathcal{N}(\mathbf{x}^n|\mathbf{m}^n,\mathbf{V}^n) \\
\mathbf{M-step}: \Lambda^{t+1} &= \text{argmax}_\Lambda \sum_n \left\langle \ell_c(\mathbf{x}^n,\mathbf{y}^n)\right\rangle_{q_n^{t+1}} \\
\Psi^{t+1} &= \text{argmax}_\Psi \sum_n \left\langle \ell_c(\mathbf{x}^n,\mathbf{y}^n)\right\rangle_{q_n^{t+1}}
\end{aligned}$$

Inference is Linear

- Note: inference just multiplies $\mathbf{y}$ by a matrix:

$$\begin{aligned} p(\mathbf{x}|\mathbf{y}) &= \mathcal{N}(\mathbf{x}|\mathbf{m}, \mathbf{V}) \\ \mathbf{V} &= I - \Lambda^\top(\Lambda\Lambda^\top + \Psi)^{-1}\Lambda \\ &= (I + \Lambda^\top\Psi^{-1}\Lambda)^{-1} \\ \mathbf{m} &= \Lambda^\top(\Lambda\Lambda^\top + \Psi)^{-1}(\mathbf{y} - \mu) \\ &= \mathbf{V}\Lambda^\top\Psi^{-1}(\mathbf{y} - \mu) \end{aligned}$$

- Note: inference of the posterior mean is just a linear operation!

$$\mathbf{m} = \beta(\mathbf{y} - \mu)$$

  where β can be computed beforehand given the model parameters.

- Also: posterior covariance does not depend on observed data!

$$\mathrm{cov}[\mathbf{x}|\mathbf{y}] = \mathbf{V} = (I + \Lambda^\top\Psi^{-1}\Lambda)^{-1}$$

Principal Component Analysis

- In Factor Analysis, we can write the marginal density explicitly:

$$p(\mathbf{y}|\theta) = \int_{\mathbf{x}} p(\mathbf{x})p(\mathbf{y}|\mathbf{x}, \theta)d\mathbf{x} = \mathcal{N}(\mathbf{y}|\mu\ ,\ \Lambda\Lambda^\top + \Psi)$$

- Noise Ψ mut be restricted for model to be interesting. (Why?)
- In Factor Analysis the restriction is that Ψ is *diagonal* (axis-aligned).
- What if we further restrict $\Psi = \sigma^2 I$ (ie *spherical*)?
- We get the Probabilistic Principal Component Analysis (PPCA) model:

$$\begin{aligned} p(\mathbf{x}) &= \mathcal{N}(\mathbf{x}|0, I) \\ p(\mathbf{y}|\mathbf{x}, \theta) &= \mathcal{N}(\mathbf{y}|\mu + \Lambda\mathbf{x}, \sigma^2 I) \end{aligned}$$

  where μ is the mean vector,
  columns of Λ are the *principal components* (usually orthogonal),
  and σ^2 is the *global sensor noise*.

Final Algorithm: EM for Factor Analysis

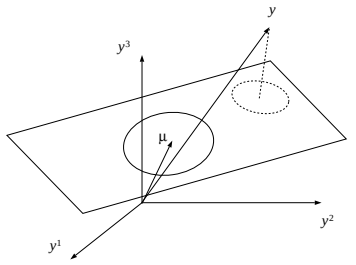
- First, set μ equal to the sample mean $(1/N)\sum_n \mathbf{y}_n$, and subtract this mean from all the data.
- Now run the following iterations:

$$\begin{aligned} \mathbf{E-step}: q^{t+1} &= p(\mathbf{x}|\mathbf{y}, \theta^t) = \mathcal{N}(\mathbf{x}^n|\mathbf{m}^n, \mathbf{V}^n) \\ \mathbf{V}^n &= (I + \Lambda^\top\Psi^{-1}\Lambda)^{-1} \\ \mathbf{m}^n &= \mathbf{V}^n\Lambda^\top\Psi^{-1}(\mathbf{y} - \mu) \\ \mathbf{M-step}: \Lambda^{t+1} &= \left(\sum_n \mathbf{y}^n\mathbf{m}^{n\top}\right)\left(\sum_n \mathbf{V}^n\right)^{-1} \\ \Psi^{t+1} &= \frac{1}{N}\mathrm{diag}\left[\sum_n \mathbf{y}^n\mathbf{y}^{n\top} + \Lambda^{t+1}\sum_n \mathbf{m}^n\mathbf{y}^{n\top}\right] \end{aligned}$$

PCA: Zero Noise Limit

- The traditional PCA model is actually a limit as $\sigma^2 \to 0$.
  The model we saw is actually called "probabilistic PCA".
- However, the ML parameters Λ^* are the same.
  The only difference is the global sensor noise σ^2.
- In the zero noise limit inference is easier: orthogonal projection.

$$\lim_{\sigma^2 \to 0} \Lambda^\top(\Lambda\Lambda^\top + \sigma^2 I)^{-1} = (\Lambda^\top\Lambda)^{-1}\Lambda^\top$$



Direct Fitting

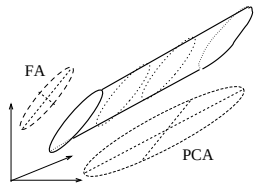
- For FA the parameters are coupled in a way that makes it impossible to solve for the ML params directly.
  We must use EM or other nonlinear optimization techniques.
- But for (P)PCA, the ML params can be solved for directly:
  The k^{th} column of Λ is the k^{th} largest eigenvalue of the sample covariance $\mathbf{S}$ times the associated eigenvector.
- The global sensor noise σ^2 is the sum of all the eigenvalues smaller than the k^{th} one.
- This technique is good for initializing FA also.
- Actually PCA is the limit as the ratio of the noise variance on the output to the prior variance on the latent variables goes to zero.
  We can either achieve this with zero noise or with infinite variance priors.

Scale Invariance in Factor Analysis

- In FA the *scale* of the data is unimportant: we can multiply $\mathbf{y}_i$ by α_i without changing anything:

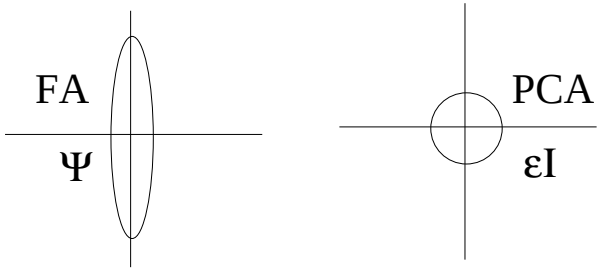
$$\begin{aligned} \mu_i &\leftarrow \alpha_i \mu_i \\ \Lambda_{ij} &\leftarrow \alpha_i \Lambda_{ij} \quad \forall j \\ \Psi_i &\leftarrow \alpha_i^2 \Psi_i \end{aligned}$$

- However, the *rotation* of the data *is* important.
- FA looks for directions of large correlation in the data, so it is not fooled by large variance noise.



Gaussians are Footballs in High-D

- Recall the intuition that Gaussians are hyperellipsoids.
- Mean == centre of football
  Eigenvectors of covarianc matrix == axes of football
  Eigenvalues == lengths of axes
- In FA our football is an axis aligned cigar.
  In PPCA our football is a sphere of radius σ^2.

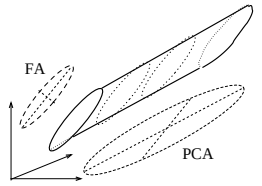


Rotational Invariance in PCA

- In PCA the *rotation* of the data is unimportant: we can multiply the data $\mathbf{y}$ by and rotation $\mathbf{Q}$ without changing anything:

$$\begin{aligned} \mu &\leftarrow \mathbf{Q}\mu \\ \Lambda &\leftarrow \mathbf{Q}\Lambda \\ \Psi &\leftarrow \text{unchanged} \end{aligned}$$

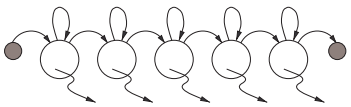
- However, the *scale* of the data *is* important.
- PCA looks for directions of large variance, so it will chase big noise directions.



Hidden Markov Models (HMMs)

Add a latent (hidden) variable x_t to improve the model.

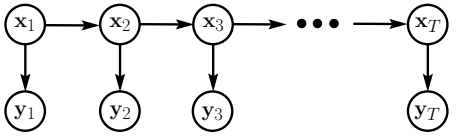
- HMM $\equiv$ " probabilistic function of a Markov chain":
  1. 1st-order Markov chain generates hidden state sequence (path):
  $$\mathsf{P}(x_{t+1} = j | x_t = i) = S_{ij} \qquad \mathsf{P}(x_1 = j) = \pi_j$$
  2. A set of output probability distributions $A_j(\cdot)$ (one per state) converts state path into sequence of observable symbols/vectors
  $$\mathsf{P}(\mathbf{y}_t = y | x_t = j) = A_j(y)$$



(state transition diagram)

- Even though hidden state seq. is 1st-order Markov, the output process is not Markov of *any* order [ex. 1111121111311121111131. . .]

HMM Graphical Model



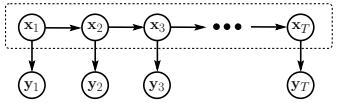
- Hidden states $\{x_t\}$, outputs $\{\mathbf{y}_t\}$
  Joint probability factorizes:
$$\mathsf{P}(\{\mathbf{x}\}, \{\mathbf{y}\}) = \prod_{t=1}^{T} \mathsf{P}(x_t | \mathbf{x}_{t-1}) \mathsf{P}(\mathbf{y}_t | x_t)$$
$$= \pi_{\mathbf{x}_1} \prod_{t=1}^{T-1} S_{x_t, x_{t+1}} \prod_{t=1}^{T} A_{x_t}(\mathbf{y}_t)$$
- NB: Data are *not* i.i.d.
  There is no easy way to use plates to show this model. (Why?)

Applications of HMMs

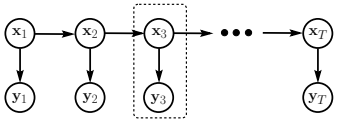
- Speech recognition.
- Language modeling.
- Information retrieval.
- Motion video analysis/tracking.
- Protein sequence and genetic sequence alignment and analysis.
- Financial time series prediction.
- . . .

Links to Other Models

- You can think of an HMM as:
  A Markov chain with stochastic measurements.



  or
  A mixture model with states coupled across time.



- The future is independent of the past given the present.
  However, conditioning on all the observations couples hidden states.

Lecture 4:

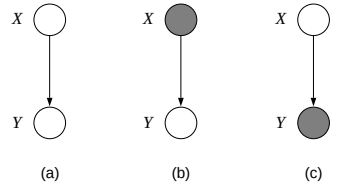
Inference in Chains and Trees, Message Passing, Belief Propagation

Sam Roweis

Friday January 28, 2005
Machine Learning Summer School

Simple Case: Bayes Rule

- For simple models, we can derive the inference formulas by hand using Bayes rule (e.g. responsibility in mixture models).



$$a) \quad p(x,y) = p(x)p(y|x)$$
$$b) \quad p(y|x)$$
$$c) \quad p(x|y) = \frac{p(x)p(y|x)}{\sum_x p(x)p(y|x)}$$

This is called "reversing the arrow".

- In general, the calculation we want to do is:

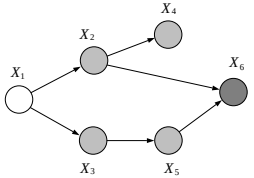
$$p(\mathbf{x}_F|\mathbf{x}_E) = \frac{\sum_{\mathbf{x}_R} p(\mathbf{x}_E, \mathbf{x}_F, \mathbf{x}_R)}{\sum_{\mathbf{x}_F, \mathbf{x}_R} p(\mathbf{x}_E, \mathbf{x}_F, \mathbf{x}_R)}$$

- Q: Can we do these sums efficiently?
  Can we avoid repeating unecessary work each time we do inference?
  A: Yes, if we exploit the factorization of the joint distribution.

Probabilistic Inference

- Partition the random variables in a domain $\mathbf{X}$ into three disjoint subsets, $\mathbf{x}_E$,$\mathbf{x}_F$,$\mathbf{x}_R$. The general *probabilistic inference* problem is to compute the posterior $p(\mathbf{x}_F|\mathbf{x}_E)$ over *query nodes* $\mathbf{x}_F$.
- This involves *conditioning* on *evidence nodes* $\mathbf{x}_E$ and *integrating (summing) out marginal nodes* $\mathbf{x}_R$.
- If the joint distribution is represented as a huge table, this is trivial: just select the appropriate indicies in the columns corresponding to $\mathbf{x}_E$ based on the values, sum over the columns corresponding to $\mathbf{x}_R$, and renormalize the resulting table over $\mathbf{x}_F$.
- If the joint is a known continuous function this can sometimes be done analytically. (e.g. Gaussian: eliminate rows/cols corresponding to $\mathbf{x}_R$; apply conditioning formulas for $p(\mathbf{x}_F|\mathbf{x}_E)$).
- But what if the joint distribution over $\mathbf{X}$ is represented by a directed or undirected graphical model?

Example



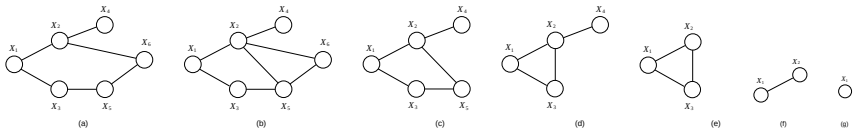
The key is to factor and then apply the distributive law.

$$p(\mathbf{x}_1|\bar{\mathbf{x}}_6) = p(\mathbf{x}_1, \bar{\mathbf{x}}_6)/p(\bar{\mathbf{x}}_6)$$
$$= p(\mathbf{x}_1, \bar{\mathbf{x}}_6)/\sum_{\mathbf{x}_1'} p(\mathbf{x}_1', \bar{\mathbf{x}}_6)$$

$$p(\mathbf{x}_1, \bar{\mathbf{x}}_6) = \sum_{\mathbf{x}_2}\sum_{\mathbf{x}_3}\sum_{\mathbf{x}_4}\sum_{\mathbf{x}_5} p(\mathbf{x}_1)p(\mathbf{x}_2|\mathbf{x}_1)p(\mathbf{x}_3|\mathbf{x}_1)p(\mathbf{x}_4|\mathbf{x}_2)p(\mathbf{x}_5|\mathbf{x}_3)p(\bar{\mathbf{x}}_6|\mathbf{x}_2, \mathbf{x}_5)$$
$$= p(\mathbf{x}_1)\sum_{\mathbf{x}_2} p(\mathbf{x}_2|\mathbf{x}_1)\sum_{\mathbf{x}_3} p(\mathbf{x}_3|\mathbf{x}_1)\sum_{\mathbf{x}_4} p(\mathbf{x}_4|\mathbf{x}_2)\sum_{\mathbf{x}_5} p(\mathbf{x}_5|\mathbf{x}_3)p(\bar{\mathbf{x}}_6|\mathbf{x}_2, \mathbf{x}_5)$$
$$= p(\mathbf{x}_1)\sum_{\mathbf{x}_2} p(\mathbf{x}_2|\mathbf{x}_1)\sum_{\mathbf{x}_3} p(\mathbf{x}_3|\mathbf{x}_1)\Phi_5(\mathbf{x}_2, \mathbf{x}_3)\sum_{\mathbf{x}_4} p(\mathbf{x}_4|\mathbf{x}_2)$$
$$= p(\mathbf{x}_1)\sum_{\mathbf{x}_2} p(\mathbf{x}_2|\mathbf{x}_1)\Phi_4(\mathbf{x}_2)\sum_{\mathbf{x}_3} p(\mathbf{x}_3|\mathbf{x}_1)\Phi_5(\mathbf{x}_2, \mathbf{x}_3)$$
$$= p(\mathbf{x}_1)\sum_{\mathbf{x}_2} p(\mathbf{x}_2|\mathbf{x}_1)\Phi_4(\mathbf{x}_2)\Phi_3(\mathbf{x}_1, \mathbf{x}_2)$$
$$= p(\mathbf{x}_1)\Phi_2(\mathbf{x}_1)$$

Single Node Posteriors

- For a single node posterior (i.e. $\mathbf{x}_F$ is a single node), there is a simple, efficient algorithm based on eliminating nodes.
- Notation: $\bar{x}_i$ is the value of evidence node x_i.
- The algorithm, called *elimination*, requires a *node ordering* to be given, which tells it which order to do the summations in.
- In this ordering, the query node must appear last.
  Graphically, we'll remove a node from the graph once we sum it out.



Elimination Algorithm

ELIMINATE($\mathcal{G}, E, F$)
  INITIALIZE($\mathcal{G}, F$)
  EVIDENCE(E)
  UPDATE($\mathcal{G}$)
  NORMALIZE(F)

INITIALIZE($\mathcal{G}, F$)
  choose an ordering I such that F appears last
  for each node X_i in $\mathcal{V}$
    place $p(x_i|x_{\pi_i})$ on the active list
  end

EVIDENCE(E)
  for each i in E
    place $\delta(x_i, \bar{x}_i)$ on the active list
  end

UPDATE($\mathcal{G}$)
  for each i in I
    find all potentials from the active list that reference x_i and remove them from the active list
    let $\phi_i(x_{T_i})$ denote the product of these potentials
    let $m_i(x_{S_i}) = \sum_{x_i} \phi_i(x_{T_i})$
    place $m_i(x_{S_i})$ on the active list
  end

NORMALIZE(F)
  $p(x_F|\bar{x}_E) \leftarrow \phi_F(x_F) / \sum_{x_F} \phi_F(x_F)$

The ELIMINATE algorithm for probabilistic inference on directed graphs.

Above, T_i denotes i plus all other nodes referenced by potentials on i

Evidence Potentials

- Elimination also uses a bookeeping trick, called *evidential functions*:

$$g(\bar{x}_i) = \sum_{x_i} g(x_i)\delta(x_i, \bar{x}_i)$$

  where $\delta(x_i, \bar{x}_i)$ is 1 if $x_i = \bar{x}_i$ and 0 otherwise.
- This trick allows us to treat conditioning in the same way as we treat marginalization. So everything boils down to doing sums:

$$p(\mathbf{x}_F|\bar{\mathbf{x}}_E) = p(\mathbf{x}_F, \bar{\mathbf{x}}_E)/p(\bar{\mathbf{x}}_E)$$
$$p(\mathbf{x}_F, \bar{\mathbf{x}}_E) = \sum_{\mathbf{x}_R}\sum_{\mathbf{x}_E} p(\mathbf{x}_F, \mathbf{x}_E, \mathbf{x}_R)\delta(\mathbf{x}_E, \bar{\mathbf{x}}_E)$$
$$p(\bar{\mathbf{x}}_E) = \sum_{\mathbf{x}_R}\sum_{\mathbf{x}_E}\sum_{\mathbf{x}_F} p(\mathbf{x}_F, \mathbf{x}_E, \mathbf{x}_R)\delta(\mathbf{x}_E, \bar{\mathbf{x}}_E)$$

- We just pick an ordering and go for it...

Algorithm Details

- At each step we are trying to remove the current variable in the elimination ordering from the distribution.
  For marginal nodes this sums them out, for evidence nodes this conditions on their observed values using the evidential functions.
- Each step in UPDATE performs a sum over a product of potential functions. Potentials can be original functions $p(x_i|\mathbf{x}_{\pi_i})$, evidential functions $\delta(x_i, \bar{x}_i)$ or intermediate potentials m_i.
- The algorithm terminates when we reach the query node, which always appears last in the ordering.
- We renormalize what we have left to get the final result:

$$p(\mathbf{x}_F|\mathbf{x}_E)$$

- For undirected models, everything is the same except the initialization phase uses the clique potentials instead of the parent-conditionals.

Marginalization without Evidence

- Marginalization of joint distributions represented by graphical models is a special case of probabilistic inference.
- To compute the marginal $p(x_i)$ of a single node, we set it to be the query node and set the evidence set to be empty.
- In directed models, we can ignore all nodes downstream from the query node, and marginalize only the part of the graph before it.
- If the node has no parents, we can read off its marginal directly.
- In undirected models, we need to do the full computation: compute $p(x_i)/Z$ using elmination and then normalize in the last step of elmination to get Z.
  (We can reuse Z later if we want to save work).

Node Elimination

- The algorithm we presented is really a way of eliminating nodes from a graph one by one. For undirected graphs:

```
foreach node x_i in ordering I:
    connect all the neighbours of x_i
    remove x_i from the graph
end
```

- The removal operation requires summing out x_i (or conditioning on observed evidence for x_i).
- Summing out x_i leaves a function involving all its previous neighbours and thus they become connected by this step.
- The original graph, augmented by all the added edges is now a *triangulated* graph. (Reminder: triangulated means that every cycle of length >3 contains a chord, ie an edge not on the cycle but between two nodes in the cycle.)

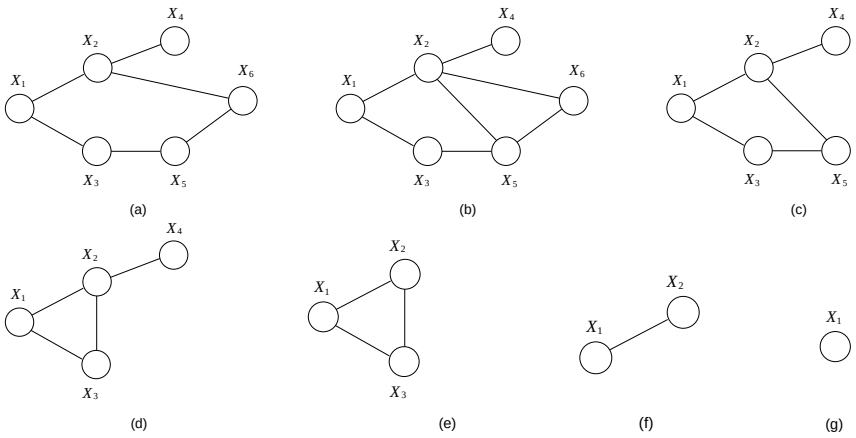
Efficiency Trick in Directed Elimination

- In directed models, we often know that a certain sum must evaluate to unity, since it is a conditional probability.
- For example, consider the term $\Phi_4(\mathbf{x}_2)$ in our six node example:

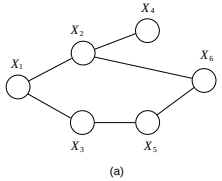
$$\Phi_4(\mathbf{x}_2) = \sum_{\mathbf{x}_4} p(\mathbf{x}_4|\mathbf{x}_2) \equiv 1$$

- We can't use this trick in undirected models, because there are no guarantees about what clique potentials sum to.

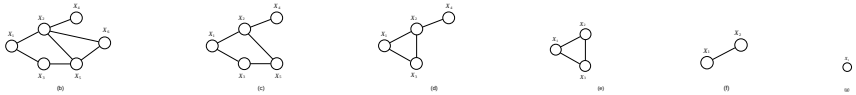
Example

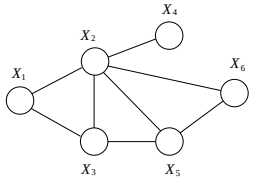


Added Edges == Triangulation



It is easy to check if a graph is triangulated in linear time. It is easy to triangulate a non-triangulated graph. But it is very hard to do so in a way that induces small clique sizes.



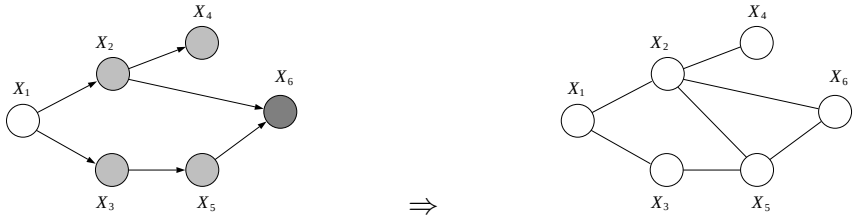


Multiple Queries

- The Elimination algorithm we described was query based: given the single node marginal to compute (the last item in the ordering), it efficiently summed out or conditioned on all other variables.
- But what if we want to do multiple inferences?
  For example, during learning, constraint satisfaction, planning.

Moralization

- For directed graphs, the parents may not be explicitly connected, but they are involved in the same potential function $p(x_i|\mathbf{x}_{\pi_i})$.
- Thus to think of Elimination as a node removal algorithm, we first must connect all the parents of every node and the drop the directions on the links.
- This step is known as "Moralization" and it is essential: since conditioning couples parents in directed models ("explaining away") we need a mechanism for respecting this when we do inference.

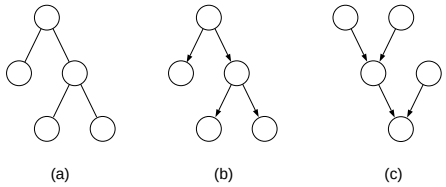


Efficient Multi-Elimination

- We could run Elimination once for each marginal, but this would be *extremely* inefficient since most of the calculations would be duplicated.
- We want an algorithm that reuses work efficiently to compute all marginals (or pairwise marginals) given evidence
- We do not want to duplicate work unecessarily.
  We want to reuse calculations as best as possible.
- This needs:
  1) A plan for which intermediate factors to compute in what order.
  2) Some storage for these intermediate factors.

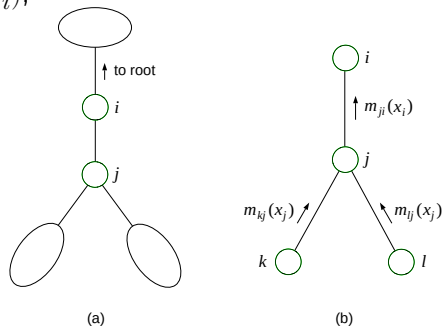
Tree-Structured Graphical Models

- For now, we will focus on tree-structured graphical models.
- Trees are an important class; they incorporate all chains (e.g. HMMs) as well.
- Exact inference on trees is the basis for the *junction tree algorithm* which solves the general exact inference problem for directed acyclic graphs and for many *approximate* algorithms which can work on intractable or cyclic graphs.
- Directed and undirected trees make exactly same conditional independence assumptions, so we cover them together.



Elimination on Trees

- Now consider eliminating node j which is followed by i in the order.
- Which nodes appear in the potential created after summing over j?
  - nothing in the subtree below j (already eliminated)
  - nothing from other subtrees, since the graph is a tree
  - only i, from ψ_{ij} which relates i and j
- Call the factor that is created $m_{ji}(x_i)$, and think of it as a *message* that j passes to i when j is eliminated.
- This message is created by summing over j the product of all earlier messages $m_{kj}(x_j)$ sent to j as well as $\psi_j^E(x_j)$ (if j is an evidence node).



Elimination on Trees

- Recall basic structure of Eliminate:
  1. Convert directed graph to undirected by moralization.
  2. Chose elimination ordering with query node last.
  3. Place all potentials on active list.
  4. Eliminate nodes by removing all relevant potentials, taking product, summing out node and placing resulting factor back onto potential list.
- What happens when the original graph is a tree?
  1. No moralization is necessary.
  2. There is a natural elimination ordering with query node as root. (Any depth first search order.)
  3. All subtrees with no evidence nodes can be ignored (since they will leave a potential of unity once they are eliminated).

Eliminate = Message Passing

- On a tree, Eliminate can be thought of as passing messages up to the query node at the root from the other nodes at the leaves or interior. Since we ignore subtrees with no evidence, observed (evidence) nodes at always at the leaves.
- The message $m_{ji}(x_i)$ is created when we sum over x_j

$$m_{ji}(x_i) = \sum_{x_j} \left(\psi^E(x_j)\psi(x_i, x_j) \prod_{k \in c(j)} m_{kj}(x_j) \right)$$

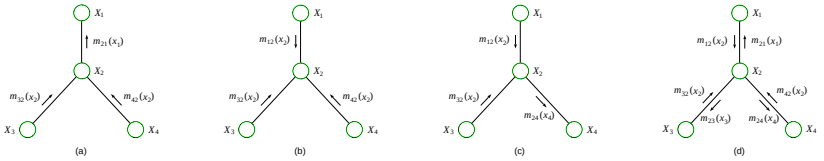
- At the final node x_f, we obtain the answer:

$$p(x_f|\bar{\mathbf{x}}_E) \propto \psi^E(x_f) \prod_{k \in c(f)} m_{kf}(x_f)$$

- If j is an evidence node, $\psi^E(x_j) = \delta(x_j, \bar{x}_j)$, else $\psi^E(x_j) = 1$.
- If j is a leaf node in the ordering, $c(j)$ is empty, otherwise $c(j)$ are the children of j in the ordering.

Messages are Reused in MultiElimination

- Consider querying x_1, x_2, x_3 and x_4 in the graph below.
- The messages needed for x_1, x_2, x_4 individually are shown (a-c).
- Also shown in (d) is the set of messages needed to compute all possible marginals over single query nodes.



- Key insight: even though the naive approach (rerun Elimination) needs to compute N^2 messages to find marginals for all N query nodes, there are only $2N$ possible messages.
- We can compute all possible messages in only double the amount of work it takes to do one query.
- Then we take the product of relevant messages to get marginals.

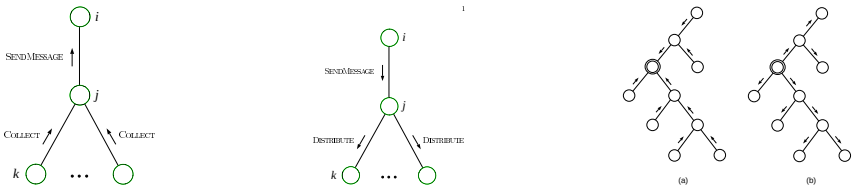
Belief Propagation (Sum-Product) Algorithm

- Choose a root node (arbitrarily or as first query node).
- If j is an evidence node, $\psi^E(x_j) = \delta(x_j, \bar{x}_j)$, else $\psi^E(x_j) = 1$.
- Pass messages from leaves up to root and then back down using:

$$m_{ji}(x_i) = \sum_{x_j} \left(\psi^E(x_j)\psi(x_i, x_j) \prod_{k \in c(j)} m_{kj}(x_j) \right)$$

- Given messages, compute marginals using:

$$p(x_i|\bar{\mathbf{x}}_E) \propto \psi^E(x_i) \prod_{k \in c(i)} m_{ki}(x_i)$$

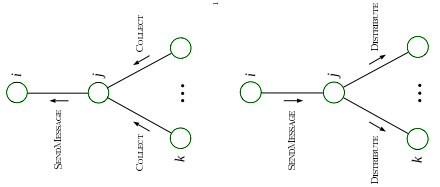


Computing All Possible Messages

- How can we compute all possible messages efficiently?
- Idea: respect the following Message-Passing-Protocol: *A node can send a message to a neighbour only when it has received messages from all its other neighbours.*
- Protocol is realizable: designate one node (arbitrarily) as the root. Collect messages inward to root then distribute back out to leaves.
- Once we have the messages, we can compute marginals using:

$$p(x_i|\bar{\mathbf{x}}_E) \propto \psi^E(x_i) \prod_{k \in c(i)} m_{ki}(x_i)$$

- Remember that the directed tree on which we pass messages might not be same directed tree we started with.
- We can also consider "synchronous" or "asynchronous" message passing nodes that respect the protocol but don't use the Collect-Distribute schedule above. (Must prove this terminates.)



SUM-PRODUCT(T, E)
  EVIDENCE(E)
  f = CHOOSEROOT($\mathcal{V}$)
  for $e \in \mathcal{N}(f)$
    COLLECT(f, e)
  for $e \in \mathcal{N}(f)$
    DISTRIBUTE(f, e)
  for $i \in \mathcal{V}$
    COMPUTEMARGINAL(i)

EVIDENCE(E)
  for $i \in E$
    $\psi^E(x_i) = \psi(x_i)\delta(x_i, \bar{x}_i)$
  for $i \notin E$
    $\psi^E(x_i) = \psi(x_i)$

COLLECT(i, j)
  for $k \in \mathcal{N}(j) \backslash i$
    COLLECT(j, k)
  SENDMESSAGE(j, i)

DISTRIBUTE(i, j)
  SENDMESSAGE(i, j)
  for $k \in \mathcal{N}(j) \backslash i$
    DISTRIBUTE(j, k)

SENDMESSAGE(j, i)
  $m_{ji}(x_i) = \sum_{x_j} (\psi^E(x_j)\psi(x_i, x_j) \prod_{k \in \mathcal{N}(j) \backslash i} m_{kj}(x_j))$

COMPUTEMARGINAL(i)
  $p(x_i) \propto \psi^E(x_i) \prod_{j \in \mathcal{N}(i)} m_{ji}(x_i)$

A sequential implementation of the SUM-PRODUCT algorithm for a tree $\mathcal{T}(\mathcal{V}, \mathcal{E})$. The algorithm works for any choice of root node, and thus we have left CHOOSEROOT unspecified. A call to COLLECT causes messages to flow inward from the leaves to the root. A subsequent call to DISTRIBUTE causes messages to flow outward from the root to the leaves. After these calls have returned, the singleton marginals can be computed locally at each node.

Computing Joint Pairwise Posteriors

- We can also easily compute the joint pairwise posterior distribution for any pair of connected nodes x_i, x_j.
- To do this, we simply take the product of all messages coming into node i (*except the message from node* j), all the messages coming into node j (*except the message from node* i) and the potentials $\psi_i(x_i), \psi_j(x_j), \psi_{ij}(x_i, x_j)$.
- The posterior is proportional to this product:

$$p(x_i, x_j|\bar{\mathbf{x}}_E) \propto \psi^E(x_i)\psi^E(x_j)\psi(x_i, x_j) \prod_{k \neq j \in c(i)} m_{ki}(x_i) \prod_{\ell \neq i \in c(j)} m_{\ell j}(x_j)$$

- These joint pairwise posteriors cover all the maximal cliques in the tree, and so those are all we need to do learning.
- Inference of other pairwise or higher order joint posteriors is possible, but more difficult.

Sum-Product, Max-Product and Semirings

- Why can we use the same trick for MAP as for marginals?
- Because multiplication distributes over $\max$ as well as sum:

$$\max(ab, ac) = a \ \max(b, c)$$

- Formally, both the "sum-product" and "max-product" pair are *commutative semirings*.
- It turns out that the "max-sum" pair is also a semiring:

$$\max(a + b, a + c) = a + \max(b, c)$$

which means we can do MAP computations in the log domain:

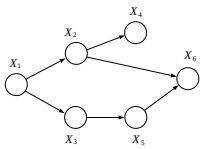
$$\max_{\mathbf{x}} p(\mathbf{x}) = \max_{\mathbf{x}} \prod_i p(x_i|\mathbf{x}_{\pi_i}) = \max_{\mathbf{x}} \log p(\mathbf{x}) = \max_{\mathbf{x}} \sum_i \log p(x_i|\mathbf{x}_{\pi_i})$$

Maximizing instead of Summing

- Elimination and Belief Propagation both summed over all possible values of the marginal (non-query, non-evidence) nodes to get a marginal probability.
- What if we wanted to *maximize* over the non-query, non-evidence nodes to find the probabilty of the single best setting consistent with any query and evidence?

$$\begin{aligned}\max_{\mathbf{x}} p(\mathbf{x}) &= \max_{\mathbf{x}_1} \max_{\mathbf{x}_2} \max_{\mathbf{x}_3} \max_{\mathbf{x}_4} \max_{\mathbf{x}_5} p(\mathbf{x}_1)p(\mathbf{x}_2|\mathbf{x}_1)p(\mathbf{x}_3|\mathbf{x}_1)p(\mathbf{x}_4|\mathbf{x}_2)p(\mathbf{x}_5|\mathbf{x}_3)p(\mathbf{x}_6|\mathbf{x}_2, \mathbf{x}_5) \\ &= \max_{\mathbf{x}_1} p(\mathbf{x}_1) \max_{\mathbf{x}_2} p(\mathbf{x}_2|\mathbf{x}_1) \max_{\mathbf{x}_3} p(\mathbf{x}_3|\mathbf{x}_1) \max_{\mathbf{x}_4} p(\mathbf{x}_4|\mathbf{x}_2) \max_{\mathbf{x}_5} p(\mathbf{x}_5|\mathbf{x}_3)p(\mathbf{x}_6|\mathbf{x}_2, \mathbf{x}_5)\end{aligned}$$

- This is known as the *maximum a-posteriori* or MAP configuration.
- It turns out that (on trees), we can use an algorithm exactly like belief-propagation to solve this problem.



Max-Product Algorithm

- Choose a root node arbitrarily.
- If j is an evidence node, $\psi^E(x_j) = \delta(x_j, \bar{x}_j)$, else $\psi^E(x_j) = 1$.
- Pass messages from leaves up to root using:

$$m_{ji}^{max}(x_i) = \max_{x_j} \left(\psi^E(x_j)\psi(x_i, x_j) \prod_{k \in c(j)} m_{kj}^{max}(x_j) \right)$$

- Remember which choice of $x_j = x_j^*$ yielded maximum.
- Given messages, compute max value using any node i:

$$\max_{\mathbf{x}} p^E(\mathbf{x}|E) = \max_{x_i} \left(\psi^E(x_i) \prod_{k \in c(i)} m_{ki}(x_i) \right)$$

- Retrace steps from root back to leaves recalling best x_j^* to get the maximizing argument (configuration) $\mathbf{x}^*$.

Efficient Probabilistic Inference

- We want to efficiently compute the posterior $p(\mathbf{x}_F|\mathbf{x}_E)$ over *query nodes* $\mathbf{x}_F$, conditioned on *evidence nodes* $\mathbf{x}_E$ while *integrating out marginal nodes* $\mathbf{x}_R$ when the joint distribution is represented by a directed or undirected graphical model.
- For some models (e.g. mixtures and factor analysis), this was an easy application of Bayes rule. For a single node posterior (i.e. $\mathbf{x}_F$ is a single node), there was a simple elimination algorithm. But it required a *node ordering* to be given, which told it which order to do the summations in and work could not be reused.
- We seek a general algorithm that is correct and takes advantage of the Markov properties of the graphical model to decompose a general calculation into a linked set of local computations.
- For tree-structured graphical models, there was the efficient Belief Propagation algorithm, but it didn't work on general DAGs.

Example: HMM Likelihood Calculation

- To evaluate the probability $\mathsf{P}(\{\mathbf{y}\})$, we want:

$$\mathsf{P}(\{\mathbf{y}\}) = \sum_{\{\mathbf{x}\}} \mathsf{P}(\{\mathbf{x}\},\{\mathbf{y}\})$$

$$\mathsf{P}(\text{observed sequence}) = \sum_{\text{all paths}} \mathsf{P}(\text{ observed outputs , state path })$$

- Looks hard! ($\#\text{paths} = \#\text{states}^T$). But joint probability factorizes:

$$\mathsf{P}(\{\mathbf{y}\}) = \sum_{\mathbf{x}_1}\sum_{\mathbf{x}_2}\cdots\sum_{\mathbf{x}_T}\prod_{t=1}^{T}\mathsf{P}(x_t|\mathbf{x}_{t-1})\mathsf{P}(\mathbf{y}_t|x_t)$$

$$= \sum_{\mathbf{x}_1}\mathsf{P}(\mathbf{x}_1)\mathsf{P}(\mathbf{y}_1|\mathbf{x}_1)\sum_{\mathbf{x}_2}\mathsf{P}(\mathbf{x}_2|\mathbf{x}_1)\mathsf{P}(\mathbf{y}_2|\mathbf{x}_2)\cdots\sum_{\mathbf{x}_T}\mathsf{P}(\mathbf{x}_T|\mathbf{x}_{T-1})\mathsf{P}(\mathbf{y}_T|\mathbf{x}_T)$$

- By moving the summations inside, we can save a lot of work.

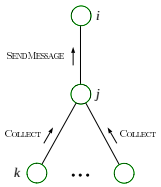
Belief Propagation (Sum-Product) Algorithm

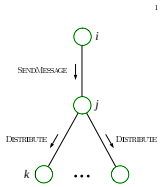
- Choose a root node arbitrarily.
- If j is an evidence node, $\psi^E(x_j) = \delta(x_j, \bar{x}_j)$, else $\psi^E(x_j) = 1$.
- Pass messages from leaves up to root and then back down using:

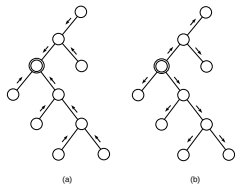
$$m_{ji}(x_i) = \sum_{x_j}\left(\psi^E(x_j)\psi(x_i,x_j)\prod_{k\in c(j)} m_{kj}(x_j)\right)$$

- Compute node marginals using the product of incoming messages:

$$p(x_i|\bar{\mathbf{x}}_E) \propto \psi^E(x_i)\prod_{k\in c(i)} m_{ki}(x_i)$$







The forward (α) recursion

- We want to compute:

$$L = \mathsf{P}(\{\mathbf{y}\}) = \sum_{\{\mathbf{x}\}} \mathsf{P}(\{\mathbf{x}\},\{\mathbf{y}\})$$

- There is a clever "forward recursion" to compute the sum efficiently.

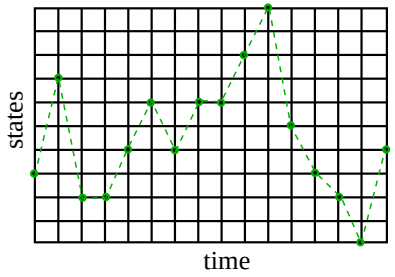
$$\alpha_j(t) = \mathsf{P}(\ \mathbf{y}_1^t\ ,\ x_t = j\ )$$
$$\alpha_j(1) = \pi_j A_j(\mathbf{y}_1)$$
$$\alpha_k(t+1) = \{\sum_j \alpha_j(t) S_{jk}\} A_k(\mathbf{y}_{t+1})$$

- Notation: $\mathbf{x}_a^b \equiv \{\mathbf{x}_a,\ldots,\mathbf{x}_b\}$; $\mathbf{y}_a^b \equiv \{\mathbf{y}_a,\ldots,\mathbf{y}_b\}$
- This enables us to easily (cheaply) compute the desired likelihood L since we know we must end in some possible state:

$$L = \sum_k \alpha_k(T)$$

HMM Inference: Bugs on a Grid

- Naive algorithm:
  1. start bug in each state at t=1 holding value 0
  2. move each bug forward in time: make copies & increment the value of each copy by transition prob. + output emission prob.
  3. go to 2 until all bugs have reached time T
  4. sum up values on all bugs



HMMs: Inference of Hidden States

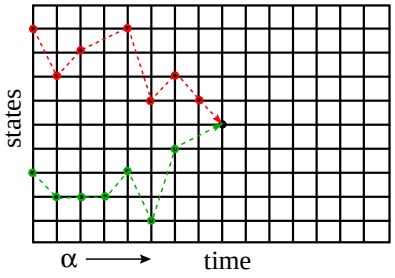
- What if we we want to estimate the hidden states given observations? To start with, let us estimate a single hidden state:

$$p(x_t|\{\mathbf{y}\}) = \gamma(x_t) = \frac{p(\{\mathbf{y}\}|x_t)p(x_t)}{p(\{\mathbf{y}\})}$$
$$= \frac{p(\mathbf{y}_1^t|x_t)p(\mathbf{y}_{t+1}^T|x_t)p(x_t)}{p(\mathbf{y}_1^T)}$$
$$= \frac{p(\mathbf{y}_1^t, x_t)p(\mathbf{y}_{t+1}^T|x_t)}{p(\mathbf{y}_1^T)}$$
$$p(x_t|\{\mathbf{y}\}) = \gamma(x_t) = \frac{\alpha(x_t)\beta(x_t)}{p(\mathbf{y}_1^T)}$$

where
$$\alpha_j(t) = p(\ \mathbf{y}_1^t\ ,\ x_t = j\ )$$
$$\beta_j(t) = p(\mathbf{y}_{t+1}^T \mid x_t = j\ )$$
$$\gamma_i(t) = p(x_t = i \mid \mathbf{y}_1^T)$$

Bugs on a Grid - Trick

- Clever recursion:
  adds a step between 2 and 3 above which says: at each node, replace all the bugs with a single bug carrying the sum of their values



- This trick is called *dynamic programming*, and can be used whenever we have a summation, search, or maximization problem that can be set up as a grid in this way. The axes of the grid don't have to be "time" and "states".

Forward-Backward Algorithm

- We compute these quantites efficiently using another recursion. Use total prob. of all paths going through state i at time t to compute the *conditional* prob. of being in state i at time t:

$$\gamma_i(t) = p(x_t = i \mid \mathbf{y}_1^T)$$
$$= \alpha_i(t)\beta_i(t)/L$$

  where we defined:

$$\beta_j(t) = p(\mathbf{y}_{t+1}^T \mid x_t = j\ )$$

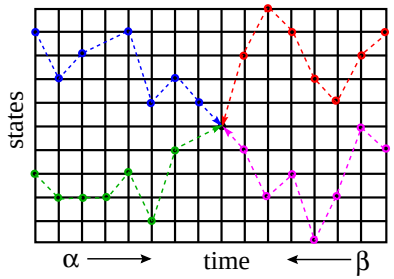
- There is also a simple recursion for $\beta_j(t)$:

$$\beta_j(t) = \sum_i S_{ji}\beta_i(t+1)A_i(\mathbf{y}_{t+1})$$
$$\beta_j(T) = 1$$

- $\alpha_i(t)$ gives total *inflow* of prob. to node (t, i)
  $\beta_i(t)$ gives total *outflow* of prob.

Forward-Backward Algorithm

- $\alpha_i(t)$ gives total *inflow* of prob. to node (t, i)
  $\beta_i(t)$ gives total *outflow* of prob.



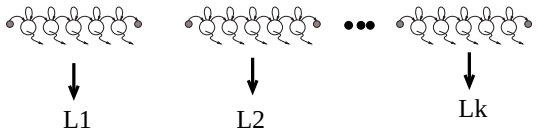
- Bugs again: we just let the bugs run forward from time 0 to t and backward from time T to t.
- In fact, we can just do one forward pass to compute all the $\alpha_i(t)$ and one backward pass to compute all the $\beta_i(t)$ and then compute any $\gamma_i(t)$ we want. Total cost is $O(K^2T)$.

Viterbi Decoding

- The numbers $\gamma_j(t)$ above gave the probability distribution over all states at any time.
- By choosing the state $\gamma_*(t)$ with the largest probability at each time, we can make a "best" state path. This is the path with the *maximum expected number of correct states*.
- But it *is not* the single path with the highest likelihood of generating the data. In fact it may be a path of prob. zero!
- To find the single best path, we do *Viterbi decoding* which is just Bellman's dynamic programming algorithm applied to this problem.
- The recursions look the same, except with $\max$ instead of $\sum$.
- Bugs once more: same trick except at each step kill all bugs but the one with the highest value at the node.

Using HMMs for Recognition

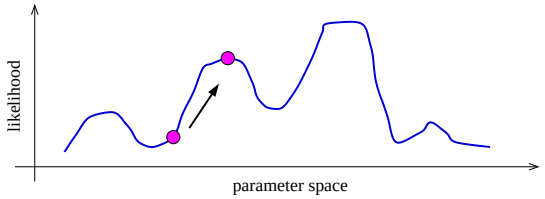
- Use many HMMs for recognition by:
  1. training one HMM for each class (requires *labelled* training data)
  2. evaluating probability of an unknown sequence under each HMM
  3. classifying unknown sequence: HMM with highest likelihood



- This requires the solution of two problems:
  1. Given model, evaluate prob. of a sequence.
     (We can do this exactly & efficiently.)
  2. Give some training sequences, estimate model parameters.
     (We can find the local maximum of parameter space nearest our starting point using Baum-Welch (EM).)

Baum-Welch Algorithm: EM Training

1. Intuition: if only we *knew* the true state path then ML parameter estimation would be trivial (MM1 on x, conditional on y).
2. But: can *estimate* state path using inference recursions.
3. *Baum-Welch algorithm* (special case of EM): estimate the states, then compute params, then re-estimate states, and so on ...
4. This works and we can *prove* that it always improves likelihood.
5. However: finding the ML parameters is NP complete, so initial conditions matter a lot and convergence is hard to tell.



Parameter Estimation using EM

- S_{ij} are transition probs; state j has output distribution $A_j(\mathbf{y})$

$$\mathsf{P}(x_{t+1}=j|x_t=i)=S_{ij} \qquad \mathsf{P}(x_1=j)=\pi_j$$
$$\mathsf{P}(\mathbf{y}_t=y|x_t=j)=A_j(y)$$

- Complete log likelihood:

$$\log p(x,y)=\log\{\pi_{\mathbf{x}_1}\prod_{t=1}^{T-1}S_{x_t,x_{t+1}}\prod_{t=1}^{T}A_{x_t}(\mathbf{y}_t)\}$$
$$=\log\{\prod_i \pi_i^{[\mathbf{x}_1^i]}\prod_{t=1}^{T-1}\prod_{ij}S_{ij}^{[x_t^i,x_{t+1}^j]}\prod_{t=1}^{T}\prod_k A_k(\mathbf{y}_t)^{[\mathbf{x}_t^k]}\}$$
$$=\sum_i[\mathbf{x}_1^i]\log\pi_i+\sum_{t=1}^{T-1}\sum_{ij}[x_t^i,x_{t+1}^j]\log S_{ij}+\sum_{t=1}^{T}\sum_k[x_t^k]\log A_k(\mathbf{y}_t)$$

where the indicator $[x_t^i]=1$ if $x_t=i$ and 0 otherwise

- For EM, we need to compute the *expected complete log likelihood*.

State expectations required from the E-Step

- The expected complete log likelihood requires
$\gamma_i(t)=<[x_t^i]>$ *and* $x_{iij}(t)=<[x_t^i,x_{t+1}^j]>$
- So in the E-step we need to compute both
$\gamma_i(t)=p(x_t=i|\{\mathbf{y}\})$ and $x_{iij}(t)=p(x_t=i,x_{t+1}=j|\{\mathbf{y}\})$.
- We already know how to compute $\gamma_i(t)$ using α and β recursions. We can compute $x_{iij}(t)$ the same way (recall BP):

$$x_{iij}(t)=p(x_{it},x_{jt+}|\{\mathbf{y}\})=p(x_{it}|\{\mathbf{y}\})p(x_{jt+}|x_{it},\{\mathbf{y}\})$$
$$=p(x_{it},y_1^t|y_{t+1}^T)p(x_{jt+}|x_{it},y_{t+1}^T)/p(y_1^t|y_{t+1}^T)$$
$$=\frac{p(x_{it},y_1^t)p(y_{t+1}^T|x_{it},y_1^t)}{p(y_1^t|y_{t+1}^T)p(y_{t+1}^T)}\frac{p(y_{t+1}^T|x_{jt+},x_{it})p(x_{jt+}|x_{it})}{p(y_{t+1}^T|x_i=t)}$$
$$=\frac{p(x_{it},y_1^t)p(y_{t+1}^T|x_{it})}{p(y_1^T)}\frac{p(y_{t+1}|x_{jt+})p(y_{t+2}^T|x_{jt+})p(x_{jt+}|x_{it})}{p(y_{t+1}^T|x_i=t)}$$
$$=\alpha_i(t)A_j(y_{t+1})S_{ij}\beta_j(t+1)/L$$

M-step: New Parameters are just Ratios of Frequency Counts

- Initial state distribution: expected #times in state i at time 1:

$$\hat{\pi}_i=\gamma_i(1)$$

- Expected #transitions from state i to j which begin at time t:

$$x_{iij}(t)=\alpha_i(t)S_{ij}A_j(\mathbf{y}_{t+1})\beta_j(t+1)/L$$

so the estimated transition probabilities are:

$$\hat{S}_{ij}=\sum_{t=1}^{T-1}x_{iij}(t)\Big/\sum_{t=1}^{T-1}\gamma_i(t)$$

- The output distributions are the expected number of times we observe a particular symbol in a particular state:

$$\hat{A}_j(y_0)=\sum_{t|\mathbf{y}_t=y_0}\gamma_j(t)\Big/\sum_{t=1}^{T}\gamma_j(t)$$

HMM Practicalities

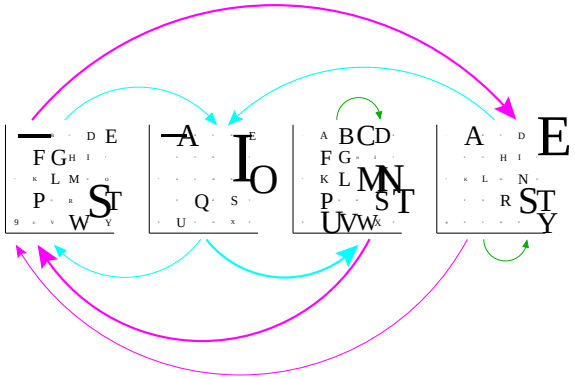
- Multiple observation sequences: can be dealt with by averaging numerators and averaging denominators in the ratios given above.
- Initialization: mixtures of Naive Bayes or mixtures of Gaussians
- Numerical scaling: the probability values that the bugs carry get tiny for big times and so can easily underflow. Good rescaling trick:

$$\rho_t=\mathsf{P}(\mathbf{y}_t|\mathbf{y}_1^{t-1}) \qquad \alpha(t)=\tilde{\alpha}(t)\prod_{t'=1}^{t}\rho_{t'}$$

or represent all probabilities as logs and use `logsum`

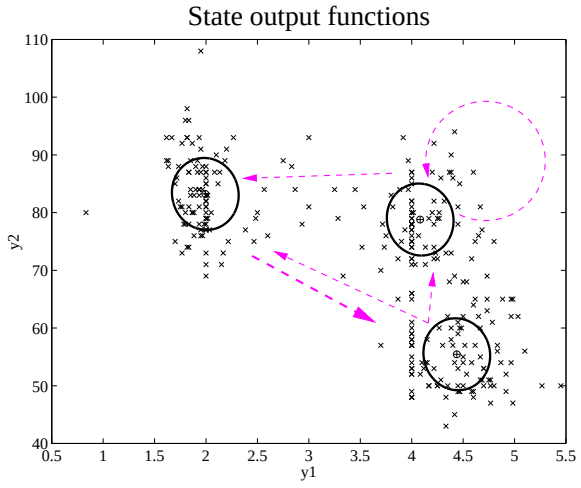
Symbol HMM Example

- Character sequences (discrete outputs)



Mixture HMM Example

- Geyser data (continuous outputs)



Summary: Probabilistic Graphical Models

- Graphical models provide a *compact factorizations* of large joint probability distributions by exploiting *conditional independencies*.
- Efficient algorithms exist for learning the parameters of a graphical model and for inferring distributions over certain variables in the model given observations of other variables.
- The simplest graphical models have only a single node and represent parametric distributions as in traditional statistics.
- The next most complex models have two nodes and represent classification, regression, clustering and latent factor models.
- Even more complex models have chain and tree structures.
- For fully observed models, maximum likelihood learning decouples across the network and each node can learn its parameters given only observations of itself and its parents.

Summary: EM and Belief Propagation

- In networks with hidden or latent variables, learning is much harder and requires inferring the distribution over the unobserved variables given the observed variables.
- Given the results of such inference, the EM algorithm can be used to update the parameters in a way that never decreases the likelihood of the training data.
- Inference in two-node models is just a simple application of Bayes' rule. This is used at test time in supervised models and at training time in unsupervised models.
- In more complex models such as chains and trees, efficient inference can be performed with the belief propagation (BP) algorithm which is the natural extension of dynamic programming to statistical models.