Algorithms for Graphical Models (AGM)

# Variable elimination

$Date: 2008/10/16 10:18:07 $

AGM-07

# In this lecture

- Variable elimination
- Cluster Forests

## Computing marginal distributions

- The basic inference problem: Given a joint distribution compute the marginal distribution for a given variable.

- One option (which is only useful conceptually) is to multiply all factors to get one (often enormous) factor and then marginalise on that factor as previously explained.

- For factored distributions it is better to *interleave* multiplication and addition, this is the basis of the *variable elimination* algorithm.

## Exploiting common factors

- Suppose we had to compute $xy + xw + xz + xu$.

- Doing so 'directly' involves 4 multiplications and 3 additions.

- Rewriting as $x(y + w + z + u)$ involves only 1 multiplication and 3 additions.

- This is the elementary arithmetic fact that variable elimination exploits.

## Principles of variable elimination

1. Replacing several factors by the single factor which is their product does not alter the distribution represented. (Multiplication is associative.)

2. To sum out several variables we can sum them out—'eliminate them'—one at a time. (Addition is associative.)

3. If a variable appears in *only one* factor then to sum it out of the distribution it is enough to sum it out of that factor. (This is the key to variable elimination . . . )

## Summing out a variable occurring in only one factor

- Let $P = \prod_{i=1}^{m} f_i$ involve variables $X_1, X_2, \ldots X_n$. Suppose wlog that we want to sum out $X_1$ and that $X_1$ only appears in factor $f_1$.

- Informally, write the desired marginal distribution as:
  $P(X_2, \ldots, X_n) = \sum_{X_1} P(X_1, X_2, \ldots X_n)$

- Claim: $P(X_2, \ldots, X_n) = (\sum_{X_1} f_1) \times (\prod_{i=2}^{m} f_i)$

## Summing out a variable occurring in only one factor (ctd)

$$
\begin{aligned}
&P(X_2 = x_2, X_3 = x_3, \ldots, X_n = x_n) \\
&\quad = \sum_{x_1 \in \mathcal{I}_{X_1}} P(X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n) \text{ [by definition]} \\
&\quad = \sum_{x_1 \in \mathcal{I}_{X_1}} \left[ f_1(X_1 = x_1, \ldots X_n = x_n) \prod_{i=2}^{m} f_i(X_2 = x_2 \ldots X_n = x_n) \right] \\
&\quad = \left[ \sum_{x_1 \in \mathcal{I}_{X_1}} f_1(X_1 = x_1, \ldots X_n = x_n) \right] \left[ \prod_{i=2}^{m} f_i(X_2 = x_2 \ldots X_n = x_n) \right]
\end{aligned}
$$

So $P(X_2, \ldots, X_n) = (\sum_{X_1} f_1) \times (\prod_{i=2}^{m} f_i)$

## How to eliminate a single variable

1. Grab all the factors containing the variable.

2. Compute their product factor and then delete them.

3. Marginalise away the variable from the product factor and add the resulting factor to the distribution.

## (Simplified) gPy source ...

```
# A method of the FR class
def eliminate_variable(self,variable):
  prod_factor = 1
  hyperedges = self._hypergraph.star(variable)
  for hyperedge in hyperedges:
    prod_factor *= self.factor(hyperedge)
    self.remove(hyperedge)
  message = prod_factor.sumout([variable])
  self *= message
```

## gPy specific stuff

- Factored distributions are `Models.FR` objects.

- Each object is determined by two attributes: a hypergraph `_hypergraph` and a dictionary `_factors`.

- The `_factors` dictionary maps each hyperedge to its corresponding factor.

- (It's a bit more complex if we allow different factors to use exactly the same variables.)

## The variable elimination algorithm

```
# A method of the FR class
def variable_elimination(self,variables,naive=True):
  """Alter a factored distribution by summing out variables"""
  for variable in variables:
    # code for when naive=False goes here
    self.eliminate_variable(variable)
```

- Not that complicated really.

- If you need to keep the original distribution, you need to store a copy first.

## Elimination orderings

- The order in which variables are summed out is called an *elimination ordering*.

- The choice of elimination ordering makes no difference to the final result (commutativity of addition).

- But makes an immense difference to the *efficiency* of the variable elimination algorithm.

- Cue `ve_demo` from `gPy.Examples`

## Visualising variable elimination: cluster forests

- For any given elimination ordering, the variable elimination algorithm can be used to generate a *cluster forest*.

- For each variable summed out, there is a '`prod_factor`' (see slide 9), whose variables we will call `prod_factor.variables()`. These sets of variables ('clusters') are the nodes of the cluster forest.

- There is an arrow from `prod_factor_1.variables()` to `prod_factor2.variables()` if the '`message`' (see slide 9 again) produced by `prod_factor_1` is one of the products in `prod_factor2`.

## Properties of cluster forests

- Cue `cluster_tree` from `gPy.Examples`.

- A *forest* is a collection of trees. A *tree* is a graph with no cycles.

- Since (1) each cluster produces exactly one message and (2) a message is deleted once it is used in another cluster ...

- ... this is enough to ensure that the cluster forest is indeed a forest.

## Cluster forests and hypergraphs

- Clusters are, of course, nothing more than hyperedges.

- The nodes of a cluster forest thus form a hypergraph—generally with a lot of redundancy.

- Hypergraphs whose hyperedges form the nodes of forests will prove to be the key data structure for probabilistic inference in factored distributions.

# Inference with evidence

- The general inference problem: Given a joint distribution **and some observed evidence** compute the marginal distribution for a given variable.

- How to compute
  $P(Cancer|XRay = abnormal, Smoking = absent)$?

## Delaying normalisation

By definition we have:

$$
\begin{aligned}
& P(L, A, T, X, D, B, S, E | X = abnormal, S = absent) \\
= \quad & \frac{P(L, A, T, X = abnormal, D, B, S = absent, E)}{P(X = abnormal, S = absent)}
\end{aligned}
$$

So can work with

$$P(L, A, T, X = abnormal, D, B, S = absent, E)$$

and delay normalising until the last moment.

## Setting zeroes

- Imagine that $P(L, A, T, X = abnormal, D, B, S = absent, E)$ were represented by one (enormous) factor.

- This factor has the same value as $P(L, A, T, X, D, B, S, E)$ for those rows where $X = abnormal$ and $S = absent$.

- But has a value of zero for those rows where it is not the case that $X = abnormal$ and $S = absent$.

- Rows for instantiations which contradict the evidence have a value of 0.

## Setting zeroes in factors

- We can define the same (unnormalised) distribution by altering the original factors:

- In each factor just set any row corresponding to an instantiation contradicting the evidence to zero and leave the other rows as they were.

- Then just run variable elimination as before.

## Avoiding pointless multiplication

- Multiplying by zero is basically a waste of time.

- So it's a bit neater to simply delete any rows which are inconsistent with the evidence. Cue `cond_demo` from `gPy.Examples`

- This amounts to not even acknowledging the existence of values which contradict the evidence!

- This is informal—in the real conditional distribution values don't just disappear—but is better algorithmically.

## Variable elimination with evidence

- Cue `ve_demo2` from `gPy.Examples`

## gPy specifics

If you want to recover the original unconditioned distribution you need to save a copy first, since conditioning alters the distribution to which it is applied:

```
from Examples import asia
as = asia.copy(copy_domain=True)
asia.condition({'Bronchitis':['absent'],'XRay':['normal']})
print as
print '***********'
print asia
```

## On not being naïve

- The variable elimination algorithm presented here is overly simple, since it does not take advantage of any (conditional) independence relations.

- Suppose we want the marginal distribution of $X$.

- If some factor contains only variables independent of $X$ (perhaps because we have conditioned on some evidence) then just delete that factor.

- It's not too difficult to concoct scenarios where this gives you a massive speed-up. See the assessment from 06-07.