In the preceding example, we saw one approach that works well when we have binary variables. In general networks, there are two common approaches to deal with this issue. The first approach is to modify the gradient ascent procedure we use (for example, conjugate gradient) to respect these constraints. First, we must project each gradient vector onto the hyperplane that satisfies the linear constraints on the parameters; this step is fairly straightforward (see exercise 19.6). Second, we must ensure that parameters are nonnegative; this requires restricting possible steps to avoid stepping out of the allowed bounds.

reparameterization

The second approach is to *reparameterize* the problem. Suppose we introduce new parameters $\lambda_{x|\boldsymbol{u}}$, and define

$$P(x \mid \boldsymbol{u}) = \frac{e^{\lambda_{x|\boldsymbol{u}}}}{\sum_{x' \in Val(X)} e^{\lambda_{x'|\boldsymbol{u}}}}, \tag{19.3}$$

for each X and its parents $\boldsymbol{U}$. Now, any choice of values for the λ parameters will lead to legal conditional probabilities. We can compute the gradient of the log-likelihood with respect to the λ parameters using the chain rule of partial derivatives, and then use standard (unmodified) conjugate gradient ascent procedure. See exercise 19.7.

Lagrange multipliers

Another way of dealing with the constraints implied by conditional probabilities is to use the method of *Lagrange multipliers*, reviewed in appendix A.5.3. Applying this method to the optimization of the log-likelihood leads to the method we discuss in the next section, and we defer this discussion; see also exercise 19.8.

Having dealt with this subtlety, we can now apply any gradient ascent procedure to find a local maximum of the likelihood function. As discussed, in most missing value problems, the likelihood function has many local maxima. Unfortunately, gradient ascent procedures are guaranteed to achieve only a local maximum of the function. Many of the techniques we discussed earlier in the book can be used to avoid local maxima and increase our chances of finding a global maximum, or at least a better local maximum: the general-purpose methods of appendix A.4.2, such as multiple random starting points, or applying random perturbations to convergence points; and the more specialized data perturbation methods of algorithm 18.1.

19.2.2 Expectation Maximization (EM)

expectation maximization

An alternative algorithm for optimizing a likelihood function is the *expectation maximization* algorithm. Unlike gradient ascent, EM is not a general-purpose algorithm for nonlinear function optimization. Rather, it is tailored specifically to optimizing likelihood functions, attempting to build on the tools we had for solving the problem with complete data.

19.2.2.1 Intuition

Recall that when learning from complete data, we can collect sufficient statistics for each CPD. We can then estimate parameters that maximize the likelihood with respect to these statistics. As we saw, in the case of missing data, we do not have access to the full sufficient statistics. Thus, we cannot use the same strategy for our problem. For example, in a simple $X \rightarrow Y$ network, if we see the training instance $\langle ?, y^1 \rangle$, then we do not know whether to count this instance toward the count $M[x^1, y^1]$ or toward the count $M[x^0, y^1]$.

A simple approach is to "fill in" the missing values arbitrarily. For example, there are strategies that fill in missing values with "default values" (say *false*) or by randomly choosing a value. Once we fill in all the missing values, we can use standard, complete data learning procedure. Such approaches are called called *data imputation* methods in statistics.

data imputation

The problem with such an approach is that the procedure we use for filling in the missing values introduces a bias that will be reflected in the parameters we learn. For example, if we fill all missing values with *false*, then our estimate will be skewed toward higher (conditional) probability of *false*. Similarly, if we use a randomized procedure for filling in values, then the probabilities we estimate will be skewed toward the distribution from which we sample missing values. This might be better than a skew toward one value, but it still presents a problem. Moreover, when we consider learning with hidden variables, it is clear that an imputation procedure will not help us. The values we fill in for the hidden variable are conditionally independent from the values of the other variables, and thus, using the imputed values, we will not learn any dependencies between the hidden variable and the other variables in the network.

A different approach to filling in data takes the perspective that, when learning with missing data, we are actually trying to solve two problems at once: learning the parameters, and hypothesizing values for the unobserved variables in each of the data cases. Each of these tasks is fairly easy when we have the solution to the other. Given complete data, we have the statistics, and we can estimate parameters using the MLE formulas we discussed in chapter 17. Conversely, given a choice of parameters, we can use probabilistic inference to hypothesize the likely values (or the distribution over possible values) for unobserved variables. Unfortunately, because we have neither, the problem is difficult.

The EM algorithm solves this "chicken and egg" problem using a bootstrap approach. We start out with some arbitrary starting point. This can be either a choice of parameters, or some initial assignment to the hidden variables; these assignments can be either random, or selected using some heuristic approach. Assuming, for concreteness, that we begin with a parameter assignment, the algorithm then repeats two steps. First, we use our current parameters to *complete* the data, using probabilistic inference. We then treat the completed data as if it were observed and learn a new set of parameters.

data completion

More precisely, suppose we have a guess $\boldsymbol{\theta}^0$ about the parameters of the network. The resulting model defines a joint distribution over all the variables in the domain. Given a partial instance, we can compute the posterior (using our putative parameters) over all possible assignments to the missing values in that instance. The EM algorithm uses this probabilistic completion of the different data instances to estimate the *expected* value of the sufficient statistics. It then finds the parameters $\boldsymbol{\theta}^1$ that maximize the likelihood with respect to these statistics.

Somewhat surprisingly, this sequence of steps provably improves our parameters. In fact, as we will prove formally, unless our parameters have not changed due to these steps (such that $\boldsymbol{\theta}^0 = \boldsymbol{\theta}^1$), our new parameters $\boldsymbol{\theta}^1$ necessarily have a higher likelihood than $\boldsymbol{\theta}^0$. But now we can iteratively repeat this process, using $\boldsymbol{\theta}^1$ as our new starting point. Each of these operations can be thought of as taking an "uphill" step in our search space. More precisely, we will show (under very benign assumptions) that: each iteration is guaranteed to improve the log-likelihood function; that this process is guaranteed to converge; and that the convergence point is a fixed point of the likelihood function, which is essentially always a local maximum. Thus, the guarantees of the EM algorithm are similar to those of gradient ascent.

19.2.2.2 An Example

We start with a simple example to clarify the concepts. Consider the simple network shown in figure 19.6. In the fully observable case, our maximum likelihood parameter estimate for the parameter $\hat{\theta}_{d^1|c^0}$ is:

$$\hat{\theta}_{d^1|c^0} = \frac{M[d^1, c^0]}{M[c^0]} = \frac{\sum_{m=1}^{M} \mathbf{1}\{\xi[m]\langle D, C\rangle = \langle d^1, c^0\rangle\}}{\sum_{m=1}^{M} \mathbf{1}\{\xi[m]\langle C\rangle = c^0\}},$$

where $\xi[m]$ is the m'th training example. In the fully observable case, we knew exactly whether the indicator variables were 0 or 1. Now, however, we do not have complete data cases, so we no longer know the value of the indicator variables.

Consider a partially specified data case $\boldsymbol{o} = \langle a^1, ?, ?, d^0\rangle$. There are four possible instantiations to the missing variables B, C which could have given rise to this partial data case: $\langle b^1, c^1\rangle$, $\langle b^1, c^0\rangle$, $\langle b^0, c^1\rangle$, $\langle b^0, c^0\rangle$. We do not know which of them is true, or even which of them is more likely.

However, assume that we have some estimate $\boldsymbol{\theta}$ of the values of the parameters in the model. In this case, we can compute how likely each of these completions is, given our distribution. That is, we can define a distribution $Q(B, C) = P(B, C \mid \boldsymbol{o}, \boldsymbol{\theta})$ that induces a distribution over the four data cases. For example, if our parameters $\boldsymbol{\theta}$ are:

$$\begin{array}{llll}
\boldsymbol{\theta}_{a^1} & = 0.3 & \boldsymbol{\theta}_{b^1} & = 0.9 \\
\boldsymbol{\theta}_{d^1|c^0} & = 0.1 & \boldsymbol{\theta}_{d^1|c^1} & = 0.8 \\
\boldsymbol{\theta}_{c^1|a^0,b^0} & = 0.83 & \boldsymbol{\theta}_{c^1|a^1,b^0} & = 0.6 \\
\boldsymbol{\theta}_{c^1|a^0,b^1} & = 0.09 & \boldsymbol{\theta}_{c^1|a^1,b^1} & = 0.2,
\end{array}$$

then $Q(B, C) = P(B, C \mid a^1, d^0, \boldsymbol{\theta})$ is defined as:

$$\begin{aligned}
Q(\langle b^1, c^1\rangle) &= 0.3 \cdot 0.9 \cdot 0.2 \cdot 0.2/0.2196 = 0.0492 \\
Q(\langle b^1, c^0\rangle) &= 0.3 \cdot 0.9 \cdot 0.8 \cdot 0.9/0.2196 = 0.8852 \\
Q(\langle b^0, c^1\rangle) &= 0.3 \cdot 0.1 \cdot 0.6 \cdot 0.2/0.2196 = 0.0164 \\
Q(\langle b^0, c^0\rangle) &= 0.3 \cdot 0.1 \cdot 0.4 \cdot 0.9/0.2196 = 0.0492,
\end{aligned}$$

where 0.2196 is a normalizing factor, equal to $P(a^1, d^0 \mid \boldsymbol{\theta})$.

If we have another example $\boldsymbol{o}' = \langle ?, b^1, ?, d^1\rangle$. Then $Q'(A, C) = P(A, C \mid b^1, d^1, \boldsymbol{\theta})$ is defined as:

$$\begin{aligned}
Q'(\langle a^1, c^1\rangle) &= 0.3 \cdot 0.9 \cdot 0.2 \cdot 0.8/0.1675 = 0.2579 \\
Q'(\langle a^1, c^0\rangle) &= 0.3 \cdot 0.9 \cdot 0.8 \cdot 0.1/0.1675 = 0.1290 \\
Q'(\langle a^0, c^1\rangle) &= 0.7 \cdot 0.9 \cdot 0.09 \cdot 0.8/0.1675 = 0.2708 \\
Q'(\langle a^0, c^0\rangle) &= 0.7 \cdot 0.9 \cdot 0.91 \cdot 0.1/0.1675 = 0.3423.
\end{aligned}$$

weighted data instances

Intuitively, now that we have estimates for how likely each of the cases is, we can treat these estimates as truth. That is, we view our partially observed data case $\langle a^1, ?, ?, d^0\rangle$ as consisting of four complete data cases, each of which has some *weight* lower than 1. The weights correspond to our estimate, based on our current parameters, on how likely is this particular completion of the partial instance. (This approach is somewhat reminiscent of the weighted particles in the likelihood weighting algorithm.) Importantly, as we will discuss, we do

not usually explicitly generate these completed data cases; however, this perspective is the basis for the more sophisticated methods.

More generally, let $\boldsymbol{H}[m]$ denote the variables whose values are missing in the data instance $\boldsymbol{o}[m]$. We now have a data set $\mathcal{D}^+$ consisting of

$$\cup_m \{\langle \boldsymbol{o}[m], \boldsymbol{h}[m]\rangle \ : \ \boldsymbol{h}[m] \in Val(\boldsymbol{H}[m])\},$$

where each data case $\langle \boldsymbol{o}[m], \boldsymbol{h}[m]\rangle$ has weight $Q(\boldsymbol{h}[m]) = P(\boldsymbol{h}[m] \mid \boldsymbol{o}[m], \boldsymbol{\theta})$.

We can now do standard maximum likelihood estimation using these completed data cases. We compute the *expected sufficient statistics*:

expected sufficient statistics

$$\bar{M}_{\boldsymbol{\theta}}[\boldsymbol{y}] = \sum_{m=1}^{M} \sum_{\boldsymbol{h}[m] \in Val(\boldsymbol{H}[m])} Q(\boldsymbol{h}[m]) \boldsymbol{1}\{\xi[m]\langle \boldsymbol{Y}\rangle = \boldsymbol{y}\}.$$

We then use these expected sufficient statistics as if they were real in the MLE formula. For example:

$$\tilde{\boldsymbol{\theta}}_{d^1|c^0} = \frac{\bar{M}_{\boldsymbol{\theta}}[d^1, c^0]}{\bar{M}_{\boldsymbol{\theta}}[c^0]}.$$

In our example, suppose the data consist of the two instances $\boldsymbol{o} = \langle a^1, ?, ?, d^0\rangle$ and $\boldsymbol{o}' = \langle ?, b^1, ?, d^1\rangle$. Then, using the calculated Q and Q' from above, we have that

$$\begin{aligned} \bar{M}_{\boldsymbol{\theta}}[d^1, c^0] &= Q'(\langle a^1, c^0\rangle) + Q'(\langle a^0, c^0\rangle) \\ &= 0.1290 + 0.3423 = 0.4713 \\ \bar{M}_{\boldsymbol{\theta}}[c^0] &= Q(\langle b^1, c^0\rangle) + Q(\langle b^0, c^0\rangle) + Q'(\langle a^1, c^0\rangle) + Q'(\langle a^0, c^0\rangle) \\ &= 0.8852 + 0.0492 + 0.1290 + 0.3423 = 1.4057. \end{aligned}$$

Thus, in this example, using these particular parameters to compute expected sufficient statistics, we get

$$\tilde{\boldsymbol{\theta}}_{d^1|c^0} = \frac{0.4713}{1.4057} = 0.3353.$$

Note that this estimate is quite different from the parameter $\theta_{d^1|c^0} = 0.1$ that we used in our estimate of the expected counts. The initial parameter and the estimate are different due to the incorporation of the observations in the data.

This intuition seems nice. However, it may require an unreasonable amount of computation. To compute the expected sufficient statistics, we must sum over all the completed data cases. The number of these completed data cases is much larger than the original data set. For each $\boldsymbol{o}[m]$, the number of completions is exponential in the number of missing values. Thus, if we have more than few missing values in an instances, an implementation of this approach will not be able to finish computing the expected sufficient statistics.

Fortunately, it turns out that there is a better approach to computing the expected sufficient statistic than simply summing over all possible completions. Let us reexamine the formula for an expected sufficient statistic, for example, $\bar{M}_{\boldsymbol{\theta}}[c^1]$. We have that

$$\bar{M}_{\boldsymbol{\theta}}[c^1] = \sum_{m=1}^{M} \sum_{\boldsymbol{h}[m] \in Val(\boldsymbol{H}[m])} Q(\boldsymbol{h}[m]) \boldsymbol{1}\{\xi[m]\langle C\rangle = c^1\}.$$

Let us consider the internal summation, say for a data case $\boldsymbol{o} = \langle a^1, ?, ?, d^0 \rangle$. We have four possible completions, as before, but we are only summing over the two that are consistent with c^1, that is, $Q(b^1, c^1) + Q(b^0, c^1)$. This expression is equal to $Q(c^1) = P(c^1 \mid a^1, d^0, \boldsymbol{\theta}) = P(c^1 \mid \boldsymbol{o}[1], \boldsymbol{\theta})$. This idea clearly generalizes to our other data cases. Thus, we have that

$$\bar{M}_{\boldsymbol{\theta}}[c^1] = \sum_{m=1}^{M} P(c^1 \mid \boldsymbol{o}[m], \boldsymbol{\theta}).$$

Now, recall our formula for sufficient statistics in the fully observable case:

$$M[c^1] = \sum_{m=1}^{M} \mathbf{1}\{\xi[m]\langle C \rangle = c^1\}.$$

Our new formula is identical, except that we have substituted our indicator variable — either 0 or 1 — with a probability that is somewhere between 0 and 1. Clearly, if in a certain data case we get to observe C, the indicator variable and the probability are the same. Thus, we can view the expected sufficient statistics as filling in soft estimates for hard data when the hard data are not available.

We stress that we use *posterior* probabilities in computing expected sufficient statistics. Thus, although our choice of $\boldsymbol{\theta}$ clearly influences the result, the data also play a central role. This is in contrast to the probabilistic completion we discussed earlier that used a prior probability to fill in values, regardless of the evidence on the other variables in the same instances.

19.2.2.3 The EM Algorithm for Bayesian Networks

We now present the basic EM algorithm and describe the guarantees that it provides.

Networks with Table-CPDs Consider the application of the EM algorithm to a general Bayesian network with table-CPDs. Assume that the algorithm begins with some initial parameter assignment $\boldsymbol{\theta}^0$, which can be chosen either randomly or using some other approach. (The case where we begin with some assignment to the missing data is analogous.) The algorithm then repeatedly executes the following phases, for $t = 0, 1, \ldots$

expected sufficient statistics

Expectation (E-step): The algorithm uses the current parameters $\boldsymbol{\theta}^t$ to compute the *expected sufficient statistics*.

- For each data case $\boldsymbol{o}[m]$ and each family $X, \boldsymbol{U}$, compute the joint distribution $P(X, \boldsymbol{U} \mid \boldsymbol{o}[m], \boldsymbol{\theta}^t)$.
- Compute the expected sufficient statistics for each $x, \boldsymbol{u}$ as:

$$\bar{M}_{\boldsymbol{\theta}^t}[x, \boldsymbol{u}] = \sum_{m} P(x, \boldsymbol{u} \mid \boldsymbol{o}[m], \boldsymbol{\theta}^t).$$

E-step

This phase is called the *E-step* (*expectation step*) because the counts used in the formula are the expected sufficient statistics, where the expectation is with respect to the current set of parameters.

Algorithm 19.2 Expectation-maximization algorithm for BN with table-CPDs

```
Procedure Compute-ESS (
    G,    // Bayesian network structure over X_1, ..., X_n
    θ,    // Set of parameters for G
    D     // Partially observed data set
)
    // Initialize data structures
    for each i = 1, ..., n
      for each x_i, u_i ∈ Val(X_i, Pa_{X_i}^G)
        M̄[x_i, u_i] ← 0
    // Collect probabilities from all instances
    for each m = 1 ... M
      Run inference on ⟨G, θ⟩ using evidence o[m]
      for each i = 1, ..., n
        for each x_i, u_i ∈ Val(X_i, Pa_{X_i}^G)
          M̄[x_i, u_i] ← M̄[x_i, u_i] + P(x_i, u_i | o[m])
    return {M̄[x_i, u_i] : ∀i = 1, ..., n, ∀x_i, u_i ∈ Val(X_i, Pa_{X_i}^G)}

Procedure Expectation-Maximization (
    G,    // Bayesian network structure over X_1, ..., X_n
    θ^0,  // Initial set of parameters for G
    D     // Partially observed data set
)
    for each t = 0, 1 ..., until convergence
      // E-step
      {M̄_t[x_i, u_i]} ← Compute-ESS(G, θ^t, D)
      // M-step
      for each i = 1, ..., n
        for each x_i, u_i ∈ Val(X_i, Pa_{X_i}^G)
          θ^{t+1}_{x_i|u_i} ← M̄_t[x_i, u_i] / M̄_t[u_i]
    return θ^t
```

Maximization (M-step): Treat the expected sufficient statistics as observed, and perform maximum likelihood estimation, with respect to them, to derive a new set of parameters. In other words, set

$$\boldsymbol{\theta}^{t+1}_{x|\boldsymbol{u}} = \frac{\bar{M}_{\boldsymbol{\theta}^t}[x, \boldsymbol{u}]}{\bar{M}_{\boldsymbol{\theta}^t}[\boldsymbol{u}]}.$$

M-step

This phase is called the *M-step* (*maximization step*), because we are maximizing the likelihood relative to the expected sufficient statistics.

A formal version of the algorithm is shown fully in algorithm 19.2.

The maximization step is straightforward. The more difficult step is the expectation step. How do we compute expected sufficient statistics? We must resort to Bayesian network inference over the network $\langle \mathcal{G}, \boldsymbol{\theta}^t \rangle$. Note that, as in the case of gradient ascent, the only expected sufficient statistics that we need involve a variable and its parents. Although one can use a variety of different inference methods to perform the inference task required for the E-step, we can, as in the case of gradient ascent, use the clique tree or cluster graph algorithm. Recall that the family-preservation property guarantees that X and its parents $\boldsymbol{U}$ will be together in some cluster in the tree or graph. **Thus, once again, we can do all of the required inference for each data case using one run of message-passing calibration.**

General Exponential Family ⋆ The same idea generalizes to other distributions where the likelihood has sufficient statistics, in particular, all models in the *exponential family* (see definition 8.1). Recall that such families have a sufficient statistic function $\tau(\xi)$ that maps a complete instance to a vector of sufficient statistics. When learning parameters of such a model, we can summarize the data using the sufficient statistic function τ. For a complete data set $\mathcal{D}^+$, we define

exponential family

$$\tau(\mathcal{D}^+) = \sum_m \tau(\boldsymbol{o}[m], \boldsymbol{h}[m]).$$

We can now define the same E and M-steps described earlier for this more general case.

E-step

Expectation (*E-step*): For each data case $\boldsymbol{o}[m]$, the algorithm uses the current parameters $\boldsymbol{\theta}^t$ to define a model, and a posterior distribution:

$$Q(\boldsymbol{H}[m]) = P(\boldsymbol{H}[m] \mid \boldsymbol{o}[m], \boldsymbol{\theta}^t).$$

expected sufficient statistics

It then uses inference in this distribution to compute the *expected sufficient statistics*:

$$\boldsymbol{E}_Q[\tau(\langle \mathcal{D}, \mathcal{H} \rangle)] = \sum_m \boldsymbol{E}_Q[\tau(\boldsymbol{o}[m], \boldsymbol{h}[m])]. \tag{19.4}$$

M-step

Maximization (*M-step*): As in the case of table-CPDs, once we have the expected sufficient statistics, the algorithm treats them as if they were real and uses them as the basis for maximum likelihood estimation, using the appropriate form of the ML estimator for this family.

Convergence Results Somewhat surprisingly, this simple algorithm can be shown to have several important properties. We now state somewhat simplified versions of the relevant results, deferring a more precise statement to the next section.

The first result states that each iteration is guaranteed to improve the log-likelihood of the current set of parameters.

Theorem 19.3

During iterations of the EM procedure of algorithm 19.2, we have

$$\ell(\boldsymbol{\theta}^t : \mathcal{D}) \leq \ell(\boldsymbol{\theta}^{t+1} : \mathcal{D}).$$

Thus, the EM procedure is constantly increasing the log-likelihood objective function. Because the objective function can be shown to be bounded (under mild assumptions), this procedure is guaranteed to converge. By itself, this result does not imply that we converge to a maximum of the objective function. Indeed, this result is only "almost true":

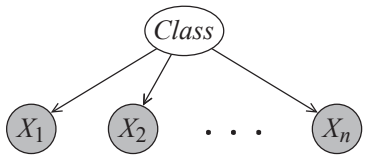

Figure 19.7 The naive Bayes clustering model. In this model each observed variables X_i is independent of the other observed variables given the value of the (unobserved) cluster variable C.

Theorem 19.4 *Suppose that $\boldsymbol{\theta}^t$ is such that $\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^t$ during EM, and $\boldsymbol{\theta}^t$ is also an interior point of the allowed parameter space. Then $\boldsymbol{\theta}^t$ is a stationary point of the log-likelihood function.*

This result shows that EM converges to a stationary point of the likelihood function. Recall that a stationary point can be a local maximum, local minimum, or a saddle point. Although it seems counterintuitive that by taking upward steps we reach a local minimum, it is possible to construct examples where EM converges to such a point. **However, nonmaximal convergence points can only be reached from very specific starting points, and are moreover not stable, since even small perturbations to the parameters are likely to move the algorithm away from this point. Thus, in practice, EM generally converges to a local maximum of the likelihood function.**

19.2.2.4 Bayesian Clustering Using EM

clustering

One important application of learning with incomplete data, and EM in particular, is to the problem of *clustering*. Here, we have a set of data points in some feature space $\boldsymbol{X}$. Let us even assume that they are fully observable. We want to classify these data points into coherent categories, that is, categories of points that seem to share similar statistical properties.

Bayesian clustering

The *Bayesian clustering* paradigm views this task as a learning problem with a single hidden variable C that denotes the category or class from which an instance comes. Each class is associated with a probability distribution over the features of the instances in the class. In most cases, we assume that the instances in each class c come from some coherent, fairly simple, distribution. In other words, we postulate a particular form for the *class-conditional distribution* $P(\boldsymbol{x} \mid c)$. For example, in the case of real-valued data, we typically assume that the class-conditional distribution is a multivariate Gaussian (see section 7.1). In discrete settings, we typically assume that the class-conditional distribution is a naive Bayes structure (section 3.1.3), where each feature is independent of the rest given the class variable. Overall, this approach views the data as coming from a *mixture distribution* and attempts to use the hidden variable to separate out the mixture into its components.

mixture distribution

naive Bayes

Suppose we consider the case of a *naive Bayes* model (figure 19.7) where the hidden class variable is the single parent of all the observed feature. In this particular learning scenario, the E-step involves computing the probability of different values of the class variables for each instance. Thus, we can think of EM as performing a soft classification of the instances, that is, each data instance belongs, to some degree, to multiple classes.

In the M-step we compute the parameters for the CPDs in the form $P(X \mid C)$ and the prior