10.2.2 Expectation Maximization for Soft Clustering

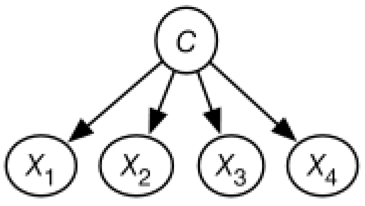
A **hidden variable** or **latent variable** is a probabilistic variable that is not observed in a data set. A Bayes classifier can be the basis for **unsupervised learning** by making the class a hidden variable.

The **expectation maximization** or **EM algorithm** can be used to learn probabilistic models with hidden variables. Combined with a **naive Bayes classifier**, it does soft clustering, similar to the k-means algorithm, but where examples are probabilistically in classes.

As in the k-means algorithm, the training examples and the number of classes, k, are given as input.

Data **Model** ⇨ **Probabilities**

X_1	X_2	X_3	X_4
t	f	t	t
f	t	t	f
f	f	t	t
…			



$P(C)$
$P(X_1 \mid C)$
$P(X_2 \mid C)$
$P(X_3 \mid C)$
$P(X_4 \mid C)$

Figure 10.4: EM algorithm: Bayes classifier with hidden class

Given the data, a naive Bayes model is constructed where there is a variable for each feature in the data and a hidden variable for the class. The class variable is the only parent of the other features. This is shown in Figure 10.4. The class variable has domain $\{1, 2, \ldots, k\}$ where k is the number of classes. The probabilities needed for this model are the probability of the class C and the probability of each feature given C. The aim of the EM algorithm is to learn probabilities that best fit the data.

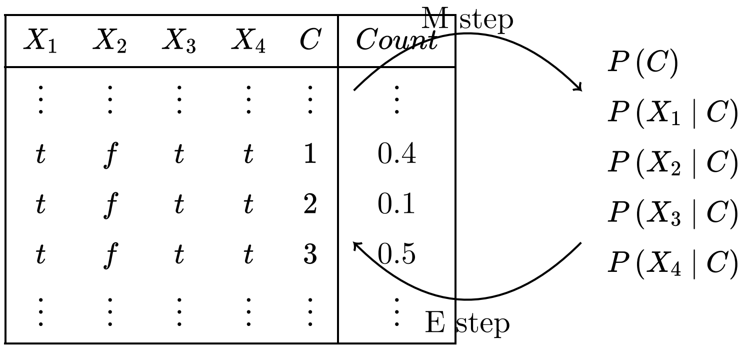
The EM algorithm conceptually augments the data with a class feature, C, and a count column. Each original example gets mapped into k augmented examples, one for each class. The counts for these examples are assigned so that they sum to 1. For example, for four features and three classes, we could have

X_1	X_2	X_3	X_4
$\vdots$	$\vdots$	$\vdots$	$\vdots$

t	f	t	t
⋮	⋮	⋮	⋮

$\longrightarrow$

X_1	X_2	X_3	X_4	C	*Count*
⋮	⋮	⋮	⋮	⋮	⋮
t	f	t	t	1	0.4
t	f	t	t	2	0.1
t	f	t	t	3	0.5
⋮	⋮	⋮	⋮	⋮	⋮



X_1	X_2	X_3	X_4	C	*Count*
⋮	⋮	⋮	⋮	⋮	⋮
t	f	t	t	1	0.4
t	f	t	t	2	0.1
t	f	t	t	3	0.5
⋮	⋮	⋮	⋮	⋮	⋮

Figure 10.5: EM algorithm for unsupervised learning

The EM algorithm repeats the two steps:

- **E step**: Update the augmented counts based on the probability distribution. For each example $\langle X_1 = v_1, \ldots, X_n = v_n \rangle$ in the original data, the count associated with $\langle X_1 = v_1, \ldots, X_n = v_n, C = c \rangle$ in the augmented data is updated to

$$P(C = c \mid X_1 = v_1, \ldots, X_n = v_n).$$

  Note that this step involves probabilistic inference. This is an **expectation** step because it computes the expected values.
- **M step**: Infer the probabilities for the model from the augmented data. Because the augmented data has values associated with all the variables, this is the same problem as learning probabilities from data in a naive Bayes classifier. This is a **maximization** step because it computes the maximum likelihood estimate or the maximum a posteriori probability (MAP) estimate of the probability.

The EM algorithm starts with random probabilities or random counts. EM will converge to a local maximum of the likelihood of the data.

This algorithm returns a probabilistic model, which is used to classify an existing or new example. An example is classified using

$$P(C=c \mid X_1=v_1,\ldots,X_n=v_n) \\ = \frac{P(C=c) * \prod_{i=1}^{n} P(X_i=v_i \mid C=c)}{\sum_{c'} P(C=c') * \prod_{i=1}^{n} P(X_i=v_i \mid C=c')} .$$

The algorithm does not need to store the augmented data, but maintains a set of **sufficient statistics**, which is enough information to compute the required probabilities. In each iteration, it sweeps through the data once to compute the sufficient statistics. The sufficient statistics for this algorithm are

- cc, the class count, a k-valued array such that $cc[c]$ is the sum of the counts of the examples in the augmented data with $class = c$
- fc, the feature count, a three-dimensional array such that $fc[i,v,c]$, for i from 1 to n, for each value v in $domain(X_i)$, and for each class c, is the sum of the counts of the augmented examples t with $X_i(t) = val$ and $class(t) = c$.

The sufficient statistics from the previous iteration are used to infer the new sufficient statistics for the next iteration. Note that cc could be computed from fc, but it is easier to maintain cc directly.

The probabilities required of the model can be computed from cc and fc:

$$P(C=c) = \frac{cc[c]}{|E|}$$

where $|E|$ is the number of examples in the original data set (which is the same as the sum of the counts in the augmented data set).

$$P(X_i=v \mid C=c) = \frac{fc[i,v,c]}{cc[c]}.$$

```
1: procedure EM($Xs, Es, k$)
2:     Inputs
3:         $Xs$ set of features, $Xs = \{X_1, \ldots, X_n\}$
4:         $Es$ set of training examples
5:         $k$ number of classes
6:     Output
7:         sufficient statistics for probabilistic model on $X$
8:     Local
9:         real $cc[c]$, $cc\_new[c]$                          ▷ old and new class count
10:        real $fc[i,v,c]$, $fc\_new[i,v,c]$                 ▷ old and new feature count
11:        real $dc$                ▷ class probability for current example and class
12:        Boolean $stable$
13:    repeat
14:        $cc\_new[c]$ and $fc\_new[i,v,c]$ initialized to be all zero
15:        for each example $\langle v_1, \ldots, v_n \rangle \in Es$ do
16:            for each $c \in [1,k]$ do
17:                $dc := P(C = c \mid X_1 = v_1, \ldots, X_n = v_n)$
18:                $cc\_new[c] := cc\_new[c] + dc$
19:                for each $i \in [1,n]$ do
20:                    $fc\_new[i, v_i, c] := fc\_new[i, v_i, c] + dc$

21:        $stable := (cc \approx cc\_new)$ and $(fc \approx fc\_new)$
22:        $cc := cc\_new$
23:        $fc := fc\_new$
24:    until $stable$
25:    return cc,fc
```

Figure 10.6: EM for unsupervised learning

Figure 10.6 gives the algorithm to compute the sufficient statistics, from which the probabilities are derived as above. Evaluating $P(C = c \mid X_1 = v_1, \ldots, X_n = v_n)$ in line 17 relies on the counts in *cc* and *fc*. This algorithm has glossed over how to initialize the counts. One way is for $P(C \mid X_1 = v_1, \ldots, X_n = v_n)$ to return a random distribution for the first iteration, so the counts come from the data. Alternatively, the counts can be assigned randomly before seeing any data. See Exercise 6.

The algoritm will eventually converge when *cc* and *fc* do not change much in an iteration. The threshold for the approximately equal in line 21 can be tuned to trade off learning time and accuracy. An alternative is to run the algorithms for a fixed number of iterations.

Notice the similarity with the *k*-means algorithm. The E step (probabilistically) assigns examples to classes, and the M step determines what the classes predict.

Example 10.10. *Consider Figure 10.5.*

When example $\langle x_1, \neg x_2, x_3, x_4 \rangle$ *is encountered in the data set, the algorithm computes*

$$\begin{aligned}
& P(C = c \mid x_1 \wedge \neg x_2 \wedge x_3 \wedge x_4) \\
\propto\ & P(X_1 = 1 \mid C = c) * P(X_2 = 0 \mid C = c) * P(X_3 = 1 \mid C = c) \\
& * P(X_4 = 1 \mid C = c) * P(C = c) \\
=\ & \frac{fc[1,1,c]}{cc[c]} * \frac{fc[2,0,c]}{cc[c]} * \frac{fc[3,1,c]}{cc[c]} * \frac{fc[4,1,c]}{cc[c]} * \frac{cc[c]}{|E|} \\
\propto\ & \frac{fc[1,1,c] * fc[2,0,c] * fc[3,1,c] * fc[4,1,c]}{cc[c]^3}
\end{aligned}$$

for each class c *and normalizes the results. Suppose the value computed for class* 1 *is 0.4, class 2 is 0.1 and for class 3 is 0.5 (as in the augmented data in Figure 10.5). Then* $cc\_new[1]$ *is incremented by* 0.4, $cc\_new[2]$ *is incremented by* 0.1, *etc. Values* $fc\_new[1,1,1]$, $fc\_new[2,0,1]$, *etc. are each incremented by* 0.4. *Next* $fc\_new[1,1,2]$, $fc\_new[2,0,2]$ *are each incremented by* 0.1, *etc.*

Note that, as long as $k > 1$, EM virtually always has multiple local maxima. In particular, any permutation of the class labels of a local maximum will also be a local maximum. To try to find a global maximum, multiple restarts can be tried, and the model with the lowest log-likelihood returned.