

# Latent class analysis

The EM algorithm for categorical data

DL Oberski & L Boeschoten

# Learning goals

- Explain how the EM algorithm works with multiple categorical indicators
- Understand and run R code implementing EM for a two-component mixture of multivariate binomials
- Explain the key assumption(s) that make LCA possible

# The model again

Mixture of  $K$  classes

$$P(\mathbf{y}) = \sum_{x=1}^K P(\mathbf{y} | X = x) P(X = x)$$

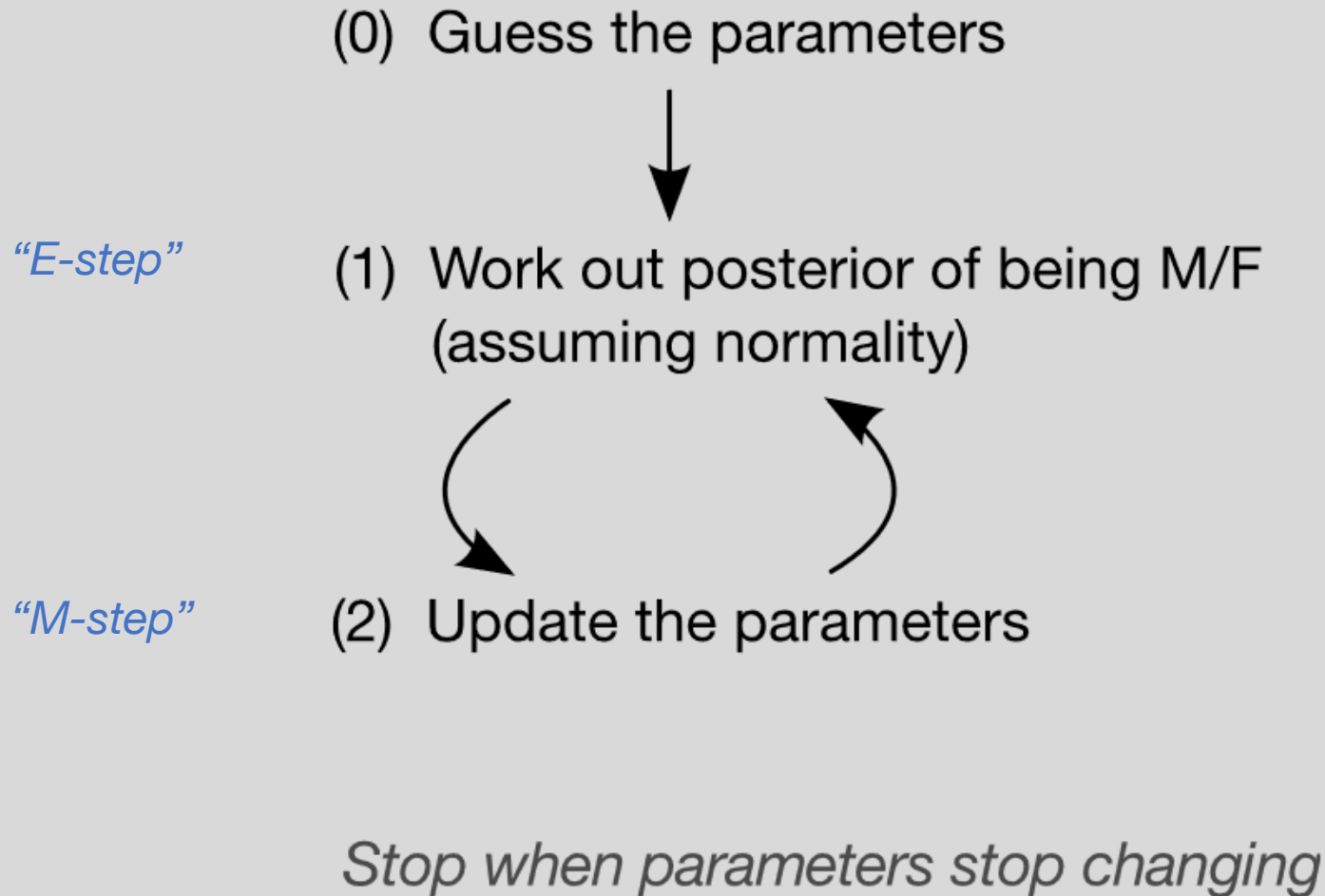
Local independence of  $p$  variables

$$P(\mathbf{y} | X = x) = \prod_{j=1}^p P(y_j | X = x)$$

Both together gives the likelihood of the observed data:

$$P(\mathbf{y}) = \sum_{x=1}^K \prod_{j=1}^p P(y_j | X = x) P(X = x)$$

# What we had before (univariate Gaussian mixture)



# M-step

- In the ***M-step***, our task is:
  - ***Given*** the posteriors (“soft guesses”) of class membership,
    1. For each indicator  $y_k$ , estimate the proportion of 1’s and 0’s in each of the classes; and
    2. Estimate the overall proportion of people in each class.
- But we already know how to do that: just calculate the mean of the (“dummy-coded”)  $y_k$ , weighted by each of the posteriors in turn.
- The overall proportion of people in each class is also easy: just the average of each posterior.

# M-step

```
# M-step for 'priors' / class size of X=2
```

```
guess_PX <- mean(post_X2)
```

```
# M-step for profiles
```

```
guess_PY.X$Y1[1] <- weighted.mean(Y1, w = (1 - post_X2))
```

```
guess_PY.X$Y1[2] <- weighted.mean(Y1, w = post_X2)
```

```
guess_PY.X$Y2[1] <- weighted.mean(Y2, w = (1 - post_X2))
```

```
guess_PY.X$Y2[2] <- weighted.mean(Y2, w = post_X2)
```

Etc.

# E-step

- In the ***E-step***, our task is:
  - ***Given*** the parameter estimates of the model,  $P(Y | X)$  and  $P(X)$ ,
  - Work out the posteriors  $P(X | Y)$  for each class
- We also know this: use the model and its **conditional independence assumption** to obtain  $P(Y|X)$  and then apply Bayes' rule.

# E-step

```
P_Y1.X1 <- dbinom(Y1, size = 1, prob = guess_PY.X$Y1[1])
```

```
P_Y1.X2 <- dbinom(Y1, size = 1, prob = guess_PY.X$Y1[2])
```

```
P_Y2.X1 <- dbinom(Y2, size = 1, prob = guess_PY.X$Y2[1])
```

```
P_Y2.X2 <- dbinom(Y2, size = 1, prob = guess_PY.X$Y2[2])
```

```
P_Y3.X1 <- dbinom(Y3, size = 1, prob = guess_PY.X$Y3[1])
```

```
P_Y3.X2 <- dbinom(Y3, size = 1, prob = guess_PY.X$Y3[2])
```



# E-step (cont.)

# Now we use the **conditional independence assumption**

```
P_Y_X1 <- P_Y1.X1 * P_Y2.X1 * P_Y3.X1
```

```
P_Y_X2 <- P_Y1.X2 * P_Y2.X2 * P_Y3.X2
```

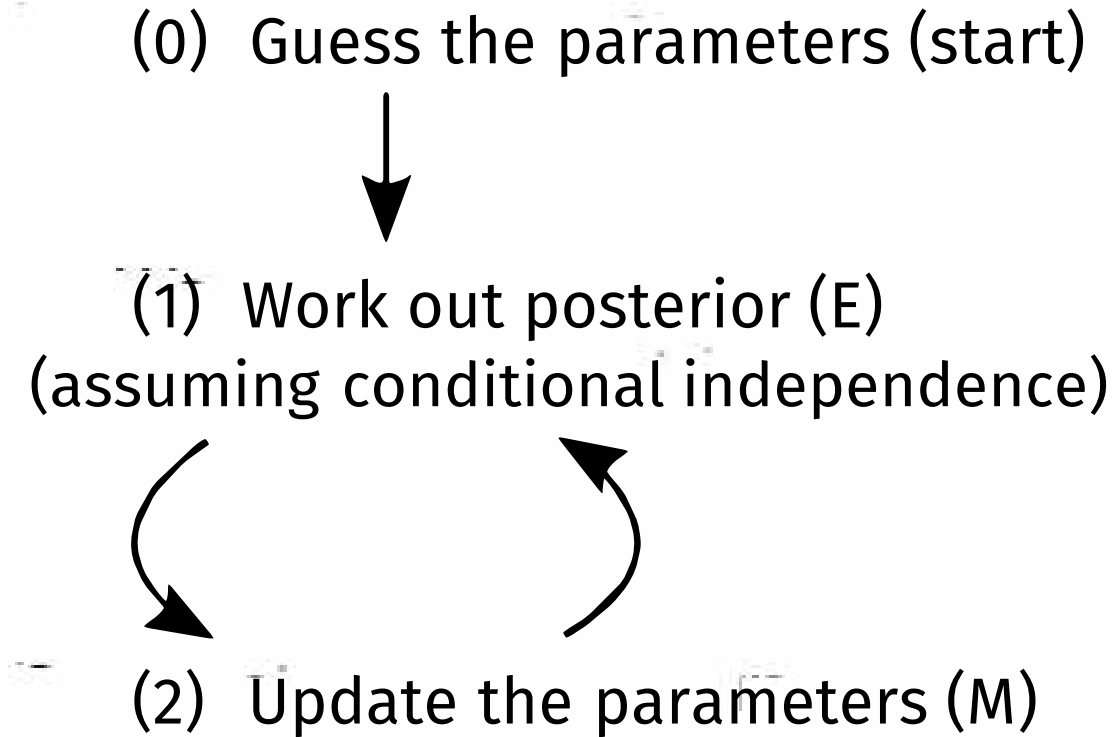
# Now we use the **mixture assumption**

```
P_Y <- (1 - guess_PX)*P_Y_X1 + guess_PX*P_Y_X2
```

# Now apply **Bayes rule** to get the posterior:

```
post_X2 <- guess_PX*P_Y_X2 / P_Y
```

# EM algorithm for categorical data



*Stop when loglikelihood stops improving*

# Loglinear LCA

$$P(A, B, C) = \sum_X \frac{e^{\lambda_{ABCX}}}{\sum_{ABC} e^{\lambda_{ABCX}}}$$

- This is just a standard loglinear model for the **complete-data table** ABCX,
- Summed over X

# Complete-data table

```
> head(df_expanded %>% arrange(A,B,C,D,E))
```

	A	B	C	D	E	Freq	patnum	X	post
1	1	1	1	1	1	34	1	0	0.1362201
2	1	1	1	1	1	34	1	1	0.8637799
3	1	1	1	1	2	2	17	0	0.8103004
4	1	1	1	1	2	2	17	1	0.1896996
5	1	1	1	2	1	0	9	0	0.1942766
6	1	1	1	2	1	0	9	1	0.8057234

# Loglinear LCA: M-step

```
formula <- Freq ~ X * (A+B+C+D+E)

fit_glm <- glm(formula,
               data = df_expanded,
               weights = post,
               family = "poisson")
```

(Venables & Ripley 2002, p.199)

# Loglinear LCA formula

`formula <- Freq ~ X * (A + B + C + D + E)`

**Note:** you could easily change the formula to include things like:

- `~A:E` (local dependence),
- `~A:E:X` (local dependence differing over classes)
- `~X:Z` (covariate/concomitant variable/grouping variable)
- `~X:Z:A` (direct effect/item bias/measurement non-invariance)
- `~X:R`, where R is a missingness indicator for the variable 'A' (MNAR)
- Etc.!

# Loglinear LCA : E-step

```
eta.X <- predict(fit_glm) # Linear predictor  
eta.X <- matrix(eta.X, nrow = n) |> t()
```

```
n.X <- sum(exp(eta.X)) # Sample size given X
```

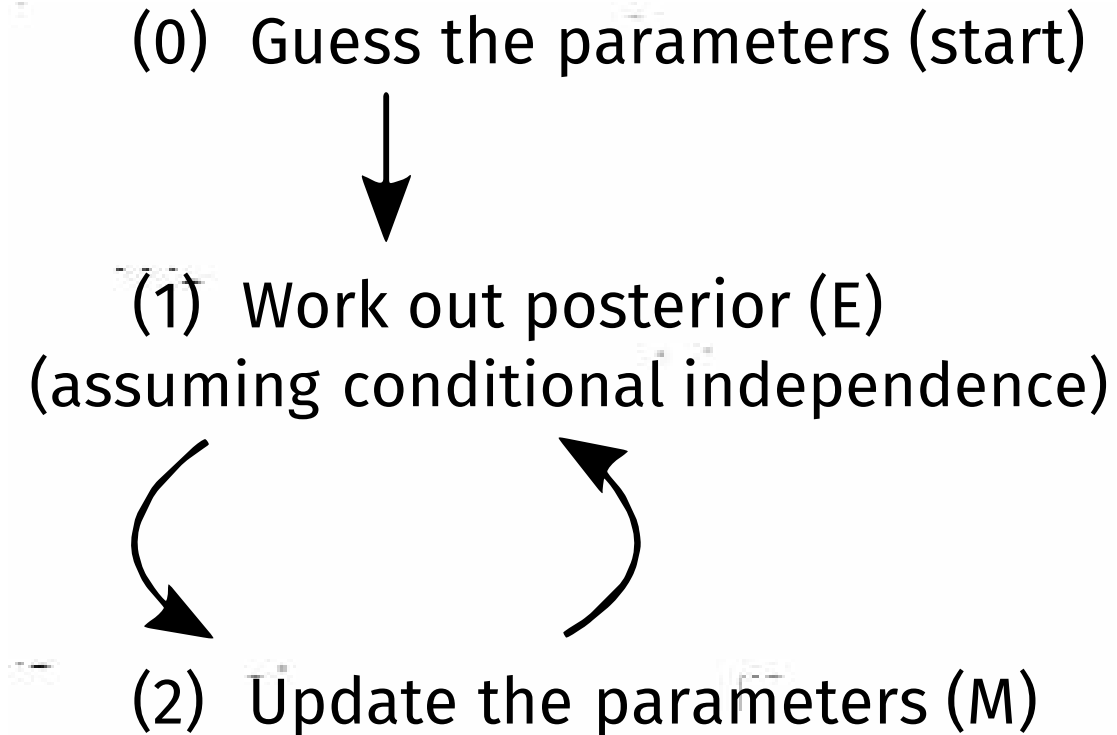
```
P_YX <- exp(eta.X)/n.X
```

```
P_Y <- colSums(P_YX)
```

```
P_X.Y <- t(P_YX) / P_Y
```

```
df_expanded$post <- as.vector(P_X.Y)
```

# EM algorithm for loglinear model



*Stop when loglikelihood stops improving*