Latent Class Analysis

In latent class analysis (LCA), the joint distribution of r *items* $Y_1 \dots Y_r$ is modelled in terms of i latent *classes*. The items are conditionally independent given the unobserved class values.

Simulation

Here we consider binary LCA models, in which the items can take one of two states, labelled "1" and "2" (Note that the `sBIC` package is capable of handling LCA models with more than two states).

In order to simulate data from an LCA model, the following parameters must be set:

1. `alpha`, a vector determining the prior probabilities of membership in each class, i.e. α_h is the probability of being in class h for $h = 1 \dots i$.
2. `P`, an $r \times i$ matrix such that P_{lh} gives the probability that $Y_l = 2$ given membership of class h.

We simulate from a model with $r = 8$ items and $i = 4$ classes. The class sizes are equal (i.e. `alpha` is uniform) and the model is "simple" in the sense that only one of the conditional probabilities P_{lh} is large (0.85) for each item and the others are small (0.1, or 0.2 depending on the item).

```
alpha = rep(0.25, 4)  ## equal

p1 = 0.85
p2 = 0.1
p3 = 0.2
P = matrix(c(
  p1,p2,p2,p2,
  p1,p3,p3,p3,
  p2,p1,p2,p2,
  p3,p1,p3,p3,
  p2,p2,p1,p2,
  p3,p3,p1,p3,
  p2,p2,p2,p1,
  p3,p3,p3,p1), nrow=8, ncol=4, byrow = TRUE)
```

The `simLCA()` function from the `poLCA` package (Drew and Linzer, 2011) simulates data from an LCA model. It requires the specification of the conditional probabilities in a different form (specifically, as a list of probability matrices) so we use a wrapper function `simLCA()`. The `simLCA()` function is designed to facilitate simulation studies, so it allows multiple simulated data sets of a given sample size to be generated. The simulated data sets are returned as a list of matrices length `nsim`.

```
simLCA <- function(alpha, ProbMat, sampleSize, nsim) {

  ## Reformat the probability Matrix to a form required by poLCA.simdata
  probs = vector("list", nrow(ProbMat))
  for (i in 1:nrow(ProbMat)) {
    probs[[i]] = cbind(ProbMat[i,], 1 - ProbMat[i,])
  }

  X = poLCA.simdata(N = sampleSize*nsim, probs = probs, P = alpha)$dat
  split(X, rep(1:nsim, each=sampleSize))
}
```

The code below generates a single data set `X` it takes the form of a 50×8 matrix with one column per item and one row per observation.

```
library(poLCA)
```

```
## Loading required package: scatterplot3d
```

```
## Warning: package 'scatterplot3d' was built under R version 3.2.5
```

```
set.seed(37421)
X = simLCA(alpha, P, sampleSize=50, nsim=1)[[1]]
head(X)
```

```
##   Y1 Y2 Y3 Y4 Y5 Y6 Y7 Y8
## 1  2  2  1  1  2  2  2  2
## 2  2  1  2  1  2  2  1  1
## 3  1  1  2  2  2  2  1  2
## 4  2  2  2  1  2  2  1  1
## 5  2  2  2  2  2  2  1  2
## 6  2  2  2  2  2  2  1  1
```

Fitting an LCA model

LCA models are fitted with the `poLCA::poLCA()` function. The `poLCA()` function uses a formula interface to determine which items are included in the model. The number of latent classes is determined by the `nclass` argument. The code below fits a 3-class model to all 8 items.

```
f = cbind(Y1, Y2, Y3, Y4, Y5, Y6, Y7, Y8) ~ 1
poLCA(f, X, nclass = 3, verbose=FALSE)
```

```
## Conditional item response (column) probabilities,
##  by outcome variable, for each class (row)
##
## $Y1
##           Pr(1) Pr(2)
## class 1:      0     1
## class 2:      0     1
## class 3:      1     0
##
## $Y2
##            Pr(1)  Pr(2)
## class 1:  0.3841 0.6159
## class 2:  0.0000 1.0000
## class 3:  0.6667 0.3333
##
## $Y3
##            Pr(1)  Pr(2)
## class 1:  0.4609 0.5391
## class 2:  0.0000 1.0000
## class 3:  0.1111 0.8889
##
```

```
## $Y4
##            Pr(1)  Pr(2)
## class 1:  0.5767 0.4233
## class 2:  0.1650 0.8350
## class 3:  0.2222 0.7778
##
## $Y5
##            Pr(1)  Pr(2)
## class 1:  0.0398 0.9602
## class 2:  1.0000 0.0000
## class 3:  0.1667 0.8333
##
## $Y6
##            Pr(1)  Pr(2)
## class 1:  0.2319 0.7681
## class 2:  1.0000 0.0000
## class 3:  0.2222 0.7778
##
## $Y7
##            Pr(1)  Pr(2)
## class 1:  0.4225 0.5775
## class 2:  0.0000 1.0000
## class 3:  0.1667 0.8333
##
## $Y8
##            Pr(1)  Pr(2)
## class 1:  0.4997 0.5003
## class 2:  0.1657 0.8343
## class 3:  0.1111 0.8889
##
## Estimated class population shares
##  0.5207 0.1193 0.36
##
## Predicted class memberships (by modal posterior prob.)
##  0.52 0.12 0.36
##
## =========================================================
## Fit for 3 latent classes:
## =========================================================
## number of observations: 50
## number of estimated parameters: 26
## residual degrees of freedom: 24
## maximum log-likelihood: -219.7884
##
## AIC(3): 491.5768
## BIC(3): 541.2894
## G^2(3): 95.87539 (Likelihood ratio/deviance statistic)
## X^2(3): 165.7594 (Chi-square goodness of fit)
##
```

Singular BIC for LCA

As with other mixture models, the learning coefficients for LCA are not known exactly, but bounds can be derived and used to obtain an approximate singular BIC. In the case of LCA models, the bounds are extremely sensitive to the prior distribution on the class probabilities α as this determines the asymptotic behaviour on the boundary $\alpha_h = 0$. The argument is developed in some detail in Drton and Plummer (2017, Section 6.3).

In the case of a Dirichlet prior with shape parameter ϕ, the bound for the learning coefficient λ_{ij} of a model with j classes embedded in a model with $i > j$ classes is given by Drton and Plummer (2017, equation 6.19)

$$\lambda_{ij} \leq \min(\lambda_{ij}^{-\phi}, \lambda_{ij}^{-r/2})$$

where $\lambda_{ij}^{-\phi}$ is defined by Drton and Plummer (2017, equation 6.15)

$$\lambda_{ij}^{-\phi} = \{jr + j - 1 + (i - j)\phi\}$$

These results suggest that $\phi \approx r/2$ should have good asymptotic properties for selecting the number of latent classes i. Tables 3 and 4 of Drton and Plummer (2017) show the results of simulation studies in support of this claim. The simulations are too extensive to reproduce in this vignette, but we step through a single example showing how the singular BIC can be recalculated for different values of ϕ.

The `sBIC::LCAs()` function creates an object of class "LCAs" representing latent class analysis models for a given number of items (`numVariables`) taking a given number of states (`numStatesForVariables`) up to a maximum number of latent states `maxNumClasses`. The resulting object of class "LCAs" is combined with the data in the `sBIC()` function, which fits the various latent class models and calculates the approximate singular BIC.

```
library(sBIC)
lcas = LCAs(maxNumClasses=6, numVariables=8, numStatesForVariables=2)
results = sBIC(X, lcas)
```

With $r = 8$ items, we choose ϕ on the order of $r/2 = 4$. The default behaviour of `sBIC()` is to use $\phi = (r+1)/2$, i.e. $\phi = 4.5$ in this example. We investigate the behaviour of the singular BIC for values around $\phi = 4$ by setting up a vector `phivec` of alternative values for ϕ.

```
phivec = (6:10)/2
```

The singular BIC can be recalculated by calling the `setPhi` member function of `lcas` to change the penalty and then calling `sBIC()` with `NULL` as the first argument. This recalculates the sBIC penalties without refitting the model to the data and so is much faster than the original call to `sBIC()`. Note that this is possible because the "LCAs" class uses call by reference semantics. The object `lcas` is altered by the initial call to `sBIC()` and on exit contains information about the fitted models.

Figure 1 shows the effect of the different values of ϕ on $\overline{BIC}_\phi$.

```
plot(1:6, results$sBIC - max(results$sBIC), type="n", xlab="number of latent classes",
     ylab=expression(bar(SBIC)[phi]))

for (i in seq_along(phivec)) {
    setPhi(lcas, phivec[i])
    results = sBIC(NULL, lcas)
    lines(1:6, results$sBIC - max(results$sBIC), pch=i, lty=i, type="b")
}
legend("topleft", pch=1:6, lty=1:6, legend=paste("phi=", phivec))
```

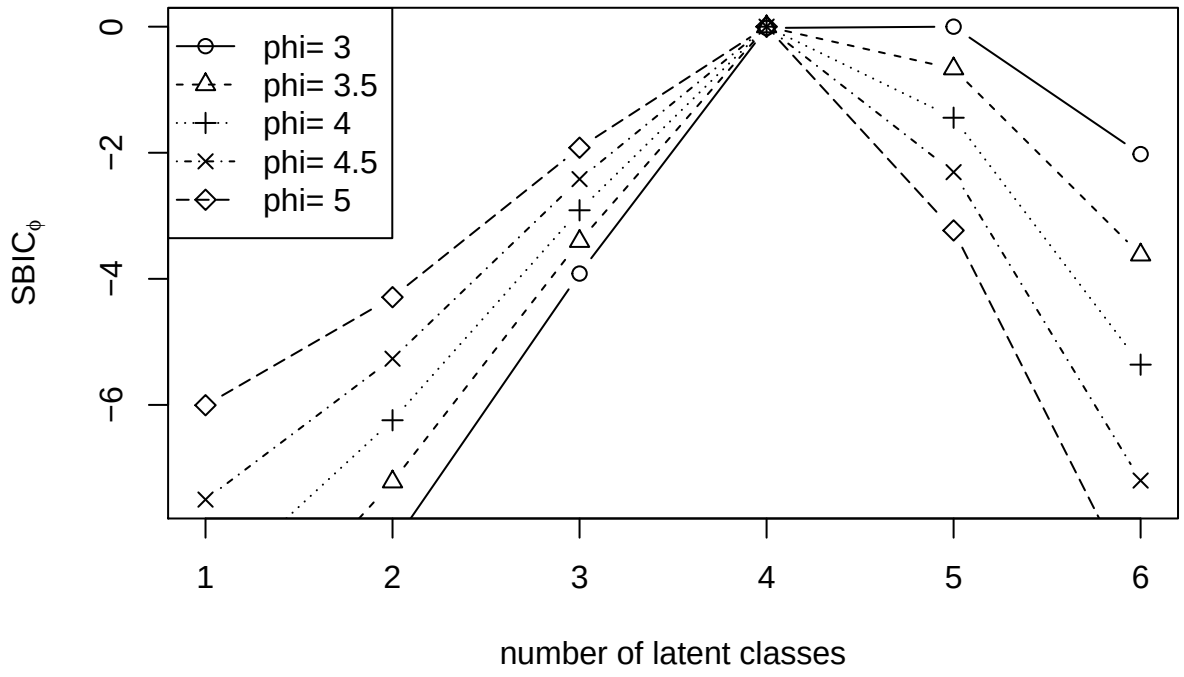

Figure 1: Influence of penalty parameter phi on sBIC for LCA

A simulation study

Further insight into the properties of $\overline{sBIC}_\phi$ can be obtained by simulation studies. The `fitBIC()` function is a convenience wrapper that calculates $\overline{sBIC}_\phi$ for various values of ϕ and chooses the optimal model according to each criterion. The return value is a vector of chosen models, indexed by the number of classes. The first element is the model chosen by BIC and the other elements are the model chosen by $\overline{sBIC}_\phi$ for the values of ϕ in the vector `phis`.

```
fitBIC <- function(X, maxNumClasses, phis) {

    lcas = LCAs(maxNumClasses, numVariables=ncol(X), numStatesForVariables=2)
    results = sBIC(X, lcas)

    nclass.bic <- which.max(results$BIC)
    nclass.sbic = numeric(length(phis))
    for (k in seq_along(phis)) {
       lcas$setPhi(phis[k])
       results = sBIC(NULL, lcas)
       nclass.sbic[k] = which.max(results$sBIC)
    }
    c(nclass.bic, nclass.sbic)
}
```

The `createTable` function calls `fitBIC` on each element of `Xlist`, a list of simulated LCA data sets and then produces a frequency table of the optimal model for BIC and for the various $\overline{sBIC}_\phi$ criteria. The function `parallel::parSapply()` is used to improve the speed of the simulations on a multi-core processor. Some code is required to set up and shutdown the cluster used for the parallel calculations, but the core of the calculations is contained in two lines of code.

```
createTable = function(Xlist, ...) {

  ## Set up cluster
  cl <- makeCluster(detectCores() - 1)
  clusterEvalQ(cl, library(sBIC))
  clusterEvalQ(cl, library(poLCA))
  clusterSetRNGStream(cl)

  ## Fit LCA models to simulated data and tabulate
  bicResults = parSapply(cl, Xlist, fitBIC, ...)
  bicTab = apply(bicResults, 1, tabulate, nbin=maxNumClasses)
  dimnames(bicTab) = list(1:maxNumClasses, c("BIC", paste0("sBIC", phis)))

  ## End cluster
  stopCluster(cl)

  bicTab
}
```

The code below simulates 100 data sets, each with a sample size of 50, and produces frequency tables of the optimal model for the different criteria.

```
library(parallel)
set.seed(1234)
Xlist = simLCA(alpha, Pr, sampleSize=50, nsim=100)
bictab = createTable(Xlist, maxNumClasses=6, phis=phivec)
knitr::kable(bictab, row.names=TRUE)
```

Table 1: Frequency distribution of the number of classes chosen by BIC and $\overline{sBIC}_\phi$ for values of ϕ around 4.

	BIC	sBIC3	sBIC3.5	sBIC4	sBIC4.5	sBIC5
1	38	0	0	0	0	0
2	42	0	0	0	1	3
3	20	3	11	16	26	34
4	0	55	72	75	70	61
5	0	38	16	9	3	2
6	0	4	1	0	0	0

Table 1 shows the resulting frequency table. Regular BIC clearly underfits the model in these simulations. Similarly $\overline{sBIC}_5$ shows signs of underfitting while $\overline{sBIC}_3$ shows overfitting.

Bibliography

- Drew A. Linzer, Jeffrey B. Lewis (2011). poLCA: An R Package for Polytomous Variable Latent Class Analysis. Journal of Statistical Software, 42(10), 1-29. URL http://www.jstatsoft.org/v42/i10/.
- Drton M. and Plummer M. (2017), A Bayesian information criterion for singular models. J. R. Statist. Soc. B; 79: 1-38.