Homework 8: Lying, Cheating, and Mixture Models

36-402, Spring 2016

Due at 11:59 pm on Wednesday, 5 April 2017

Mixture models where all the observed variables are categorical are often called **latent class models**, and the package `poLCA` provides functions for fitting them through the EM algorithm. The basic syntax is

```
poLCA(cbind(Var1,Var2,Var4)~1, data, nclass=k)
```

to fit a k-cluster model for the columns named `Var1`, `Var2` and `Var4`. Observables are assumed to be independent within each cluster. The returned object contains a lot of information, including the distribution over each observable variable for each cluster, the probability or mixing weight for each cluster, and the posterior probability for each observation having come from each cluster.

The package `poLCA` also contains a data set, `cheating`, which is the result of a survey of about 300 undergraduates on cheating. Four variables record, for each student, whether they had ever lied to get out of an exam; lied to avoid turning in a paper on time; bought a term paper or obtained a copy of an exam in advance (together, "fraud"); or copied answers on an exam from another student. The data set also records the students' GPAs, discretized into five categories. (See `help(cheating)` for details.)

1. Load the data.
    (a) (5) What are the correlations between the four forms of cheating? (Don't list them all, use a table or graph.)
    (b) (5) What fraction of students have cheated at least once? What fraction of cheaters engage in multiple forms of cheating?
    (c) (5) What fraction of students have lied to get out of an exam? What fraction of students have bought a term paper or snuck a look at an exam before taking it? What fraction of students who have lied to get out of an exam have committed fraud?
2. Fit a latent class model with two classes or clusters.
    (a) (5) For each class, what are the probabilities of each form of cheating? What is the probability of each class? (Again, use a table or graph.)

(b) (5) For each cluster, use the estimated parameters of the model to find the probability that a member of the cluster has engaged in at least one form of cheating. *Hint:* It may be easier to first find the probability that they have not cheated at all.

(c) (5) For each cluster, find the conditional probability that a member of the cluster who has engaged in at least one sort of cheating has engaged in multiple forms of cheating. Again, use the estimated parameters of the model, not a new model. *Hint:* It may be easier to first find the probability that someone has cheated *exactly* once.

(d) (5) Describe, in words, how the two classes differ from each other.

3. *Conditioning*

(a) (5) Find the probability (according to the model) that a random student has committed fraud.

(b) (10) Suppose we know that a student has lied to get out of an exam, but not whether they have engaged in any other form of cheating. Find the conditional probabilities of the student being each class. *Hint:* Bayes's rule.

(c) (10) Find the probability (according to the model) that a student who has lied to get out of an exam has also committed fraud. *Hint:* Bayes's rule, and the law of total probability.

(d) (5) Your answer for 3c should be several times larger than your answer for 3a. Explain how this is compatible with the fact that for both classes of student, lying to get out of exams is statistically independent of fraud.

4. *Cross-validation will continue until morale improves* Use five-fold cross-validation of the log-likelihood to pick the number of clusters. *Hint:* §19.4.4 of the textbook, but remember the observables here are binary, so Gaussian distributions won't work.

(a) (5) Show, in math, how to calculate the probability that a latent class model assigns to a single observation.

(b) (5) Provide comments for the following function, explaining the overall purpose of the function, what all the inputs are, what the output(s) are, how it relates to the math you just did in the previous part, and what each piece of the code does.

```
dmultbinarymix <- function(x, model, offset=1, log=FALSE) {
    x <- x-offset
    prob.matrix <- sapply(model$probs, function(mat) { mat[,2] })
    if (is.null(dim(prob.matrix))) {
        prob.matrix <- array(prob.matrix, dim=c(1,length(prob.matrix)))
    }
```

```
    class.probs <- model$P
    class.cond.prob <- function(x,c) {
        class.probs[c]*prod((prob.matrix[c,]^x)*((1-prob.matrix[c,])^(1-x)))
    }
    one.point.prob <- function(x) {
        summands <- sapply(1:length(class.probs),
                           class.cond.prob,
                           x=x)
        return(sum(summands))
    }
    probs <- apply(x, 1, one.point.prob)
    if (log) {
        return(log(probs))
    } else {
        return(probs)
    }
}
```

(c) (5) Write a function to calculate the log-likelihood that a latent class model assigns to a new data set. (This should call `dmultbinarymix`.) Check that this is working by seeing that it matches the log-likelihood returned by `poLCA` when run on the training data, and that it doesn't give exactly the same answer when you change the data.

(d) (5 Write a function to do cross-validation to pick the number of clusters. How many does it pick here?

(e) (5) Plot the cross-validated log-likelihood against the number of clusters.

Rubric (10): The text is laid out cleanly, with clear divisions between problems and sub-problems. The writing itself is well-organized, free of grammatical and other mechanical errors, and easy to follow. Questions which ask for a plot or table are answered with both the figure itself and the command (or commands) use to make the plot. Plots are carefully labeled, with informative and legible titles, axis labels, and (if called for) sub-titles and legends; they are placed near the text of the corresponding problem. All quantitative and mathematical claims are supported by appropriate derivations, included in the text, or calculations in code. Numerical results are reported to appropriate precision. Code is properly integrated with a tool like R Markdown or knitr, and both the knitted file and the source file are submitted. The code is indented, commented, and uses meaningful names. All code is relevant to the text; there are no dangling or useless commands. All parts of all problems are answered with actual coherent sentences, and never with raw computer code or its output.