Experiments with Hierarchical Multinomial Dirichlet Priors for the Conditional Probability Tables of Discrete Bayesian Networks

Michael L. Thompson

11/23/2020

Contents

Introduction

This is a brief experiment with Hierarchical Multinomial-Dirichlet priors for Conditional Probability Tables (CPTs) of discrete Bayesian Networks (i.e., Bayesian belief networks, BBN).

I demonstrate the following two approaches:

1. **`hierMD`** – The base case proposed by L. Azzimonti, G. Corani, and M. Zaffalon (*ACZ*) of Imprecise Probability Group @ IDSIA in: "Hierarchical estimation of parameters in Bayesian networks", (**I show the Stan code I adapted from their code.**)

2. **`hierMDmix`** – A mixture model of parent states that I've proposed. This approach mixes the Dirichlet parameters of each parent's state – i.e., the "local Dirichlet prior" – with those of the others to form a single parameter vector. These local-level priors each depend upon the populaton-level prior similar to how the single columnwise priors of the *ACZ* approach do.

My Conclusions (w/out having done any other testing than what's shown below):

- The more complicated mixture model gives CPT results that are less pulled towards the marginal distribution of the child node than does the *ACZ* model.
- In the resulting inferences, both approaches are an improvement over a flat prior – the traditional `BDeu` (*Bayesian Dirichlet equivalent uniform* or *Laplace smoothing*) approach.

Prepare the R Environment

Here, we load the packages we need.

```
library(magrittr) # I always use piping!
library(tidyverse) # Thank heaven & Hadley Wickham for the `tidyverse`!

# Bayesian network packages
library(bnlearn)
library(gRain)
# Implementation of **Stan** probabilistic programming language
library(rstan)
rstan_options(auto_write = TRUE)

select <- dplyr::select
```

This section includes a hidden **R** code chunk that defines functions we need (rather than `source`-ing a script). View this R Markdown document's source to see the functions.

Load the Bayesian Belief Network (BBN)

We'll use a BBN that I've generated before. The network is one of many in an ensemble of BBNs used to build a Movie Recommender System – see my presentation at the 8th Annual BayesiaLab Conference (2020) and the code and PDF of the slides at this github repository, "Bayesian Analysis".

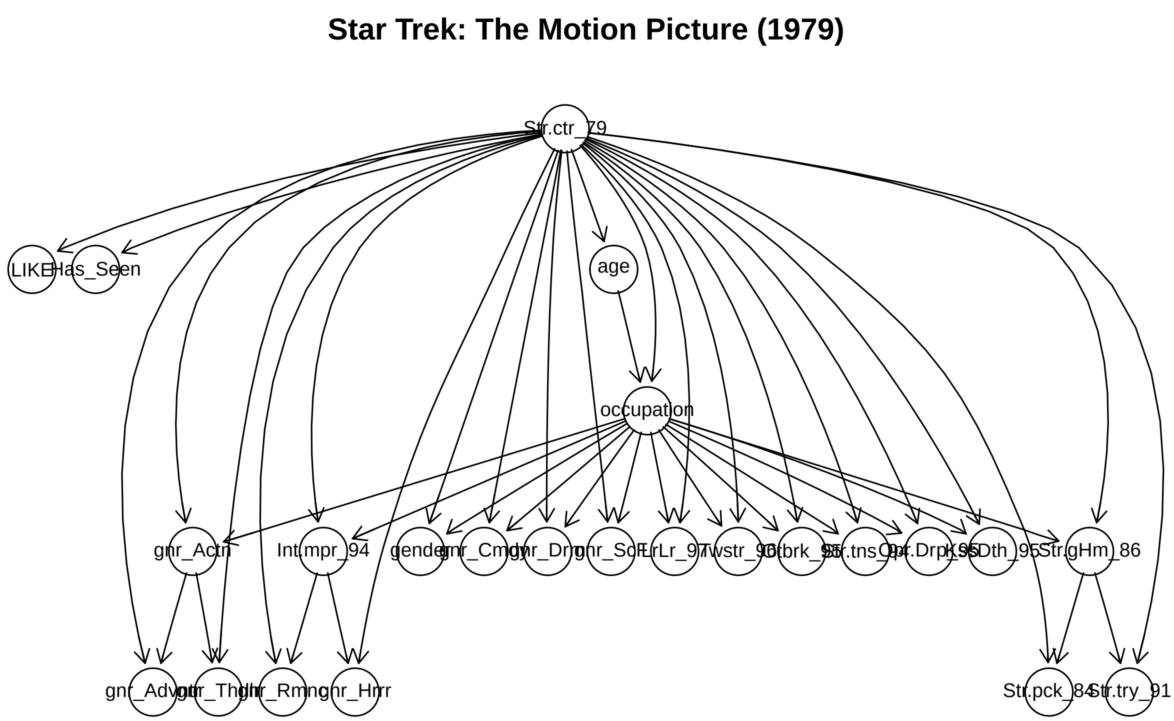
The particular BBN shown here predicts the viewer's rating for the movie "Star Trek: The Motion Picture" (1979) given any combination of viewer features – age, gender, or occupation – movie genre, and ratings of other movies. (It was based on 694 observations in the smallest version of the MovieLens dataset.)

```
# DATA LOADING ====
# Get a BBN to experiment with.
# Use it to simulate data, too.
bbn <- loadHuginNet(
  file = "bbn_StarTrek_79.net",
```

```
  description = "Predicts ratings of 'Star Trek: The Motion Picture (1979)'"
)
```

Visualize the BBN

```
# plot it
g1 <- bbn$dag  %>%
  as.bn() %>%
  graphviz.plot( render = FALSE )
g1 %>%
  plot(
    attrs = list(node=list(fontsize="32")),
    main = 'Star Trek: The Motion Picture (1979)'
  )
```



Perform Test Inferences

Just to demonstrate some of the nature of the BBN we're experimenting with, we perform a few inferences on the base BBN.

We plan on concentrating on inferences between the viewer features (`age`, `gender`, `occupation`) and the viewing (`Has_Seen`), liking (`LIKE`), and rating (`Str.ctr_79`) nodes of the BBN.

```
# The movie nodes have states that are in 1-point deviations from a viewer's
# median rating over all movies the viewer has seen. Assume dev. of zero or
# greater means viewer liked the movie.
movie_node <- "Str.ctr_79"
querygrain(bbn,"Has_Seen") %>% map(round,3)
```

```
## $Has_Seen
```

```
## Has_Seen
##   yes    no
## 0.125 0.875
```

```
querygrain(bbn,"LIKE") %>% map(round,3)
```

```
## $LIKE
## LIKE
##    yes     no unseen
##  0.065  0.060  0.875
```

```
querygrain(bbn,"LIKE", evidence = c(Has_Seen="yes") ) %>% map(round,3)
```

```
## $LIKE
## LIKE
##    yes     no unseen
##  0.521  0.479  0.000
```

```
querygrain(bbn,c("LIKE","Has_Seen"),type="conditional") %>% round(3)
```

```
##         Has_Seen
## LIKE       yes no
##   yes    0.521  0
##   no     0.479  0
##   unseen 0.000  1
```

```
querygrain(bbn,nodes = c("Has_Seen","age"),type="conditional") %>% round(3)
```

```
##         age
## Has_Seen yrs_0_15 yrs_15_25 yrs_25_33 yrs_33_44 yrs_44_110
##      yes     0.01     0.103     0.157     0.162      0.084
##      no      0.99     0.897     0.843     0.838      0.916
```

```
querygrain(
  bbn,
  nodes = c(movie_node,"age"),
  evidence = c(Has_Seen="yes"),
  type="conditional"
) %>% round(3)
```

```
##           age
## Str.ctr_79 yrs_0_15 yrs_15_25 yrs_25_33 yrs_33_44 yrs_44_110
##     r2=-3     0.167     0.116     0.026     0.001      0.002
##     r3=-2     0.167     0.116     0.175     0.118      0.293
##     r4=-1     0.167     0.345     0.275     0.264      0.235
##     r5=0      0.167     0.345     0.349     0.411      0.351
##     r6=1      0.167     0.078     0.150     0.176      0.060
##     r7=2      0.167     0.001     0.026     0.030      0.060
##     unseen    0.000     0.000     0.000     0.000      0.000
```

Generate Data

We'll also use the BBN to generate data that will serve as the bases for traditionally Laplace-smoothed parameter estimates and to see how sparsely the data lie in this high-dimensional discrete-variate space.

```
rng_seed <- 31
N    <- 1000L
df   <- bbn %>%
```

```r
simulate( nsim = N, seed = rng_seed ) %>%
as_tibble()
```

Code in the Stan Probabilistic Programming Language

We use package `rstan`, based upon the **Stan** probabilistic programming language. Here, we show the programs, one each for the hierarchical Multinomial Dirichlet approach of *ACZ*, and the mixture variant that I propose. The code below shows how straight-forward it is to implement these approaches in **Stan**. The `hierMD` code is less than 25 lines!

(Although it is possible to have `knitr` execute the code compilation directly from the **Stan** code chunks, we'll forgoe that and compile the programs from their respective files on disk in a later **R** code chunk.)

`hierMD` Stan Program

This first **Stan** program is adapted from the original by *ACZ*: I've copied the code from my file `hierMD_MLT.stan` without its banner comments acknowledging Azzimonti *et al.* – see my **Stan** files `hierMD_MLT.stan` and `hierMDmix_MLT.stan` for the full banners. This version of *ACZ*'s approach adds a proper likelihood statement and puts a prior on the magnitude of the Dirichlet parameter, `N_prior`, which in the original code was a fixed input value `s`. I've also dropped the estimation of the posterior of the marginal probability distribution parameters for the parent states (`thetaY` in the original *ACZ* code).

```stan
data {
  int<lower=2> n_st_ch; // number of child states
  int<lower=2> n_st_pr; // number of total combos of parent states
  int<lower=0> N_ch_pr[n_st_ch,n_st_pr]; // number of cases at all combos of parents & child
  vector<lower=0>[n_st_ch] alpha_0; // hyperparameter for Dirichlet priors
}
parameters {
  simplex[n_st_ch] theta[n_st_pr]; // conditional probability table parameters
  simplex[n_st_ch] alpha_norm; // population-level parameter for Dirichlet priors, normalized
  real<lower=0> N_prior; // number of cases represented by prior
}
transformed parameters {
  vector<lower=0>[n_st_ch] alpha; // population-level parameter for Dirichlet priors
  alpha = N_prior * alpha_norm;
}
model {
  alpha_norm ~ dirichlet(alpha_0); // prior
  N_prior    ~ student_t(4,1,1);   // prior
  for (i_st_pr in 1:n_st_pr){
    theta[i_st_pr]    ~ dirichlet( alpha ); // prior
    N_ch_pr[,i_st_pr] ~ multinomial( theta[i_st_pr] ); // likelihood
  }
}
```

`hierMDmix` Stan Program

And, here's the **Stan** implementation of my mixture variant, `hierMDmix`. (See file `hierMDmix_MLT.stan` for the full banner of acknowledgments to *ACZ*.)

```stan
data {
  int<lower=2> n_st_ch; // number of child states
  int<lower=2> n_st_pr; // number of total combos of parent states
  int<lower=0> N_ch_pr[n_st_ch,n_st_pr]; // number of cases at all combos of parents & child
```

```
  vector<lower=0>[n_st_ch] alpha_0; // hyperparameter for Dirichlet priors
  int <lower=1> n_parent; // number of parents
  int <lower=2> n_st_pr_i[n_parent]; // number of states for each parent
  int i_st_pr[n_st_pr,n_parent]; // state of each parent for each combo of parents
}
transformed data {
  int n_st_pr_sum; // sum of number of states for parents
  vector[n_parent] alpha_pr_mix; // Dirichlet parameters for mixture
  n_st_pr_sum  = sum(n_st_pr_i);
  alpha_pr_mix = rep_vector(1,n_parent);
}
parameters {
  simplex[n_st_ch] theta[n_st_pr];  // conditional probability table parameters
  simplex[n_st_ch] alpha_norm; // population-level parameter for Dirichlet priors, normalized
  simplex[n_st_ch] alpha_i_norm[n_st_pr_sum];// prior for the local states
  real<lower=0> N_prior;  // number of cases represented by prior
  simplex[n_parent] p_mix; // mixture probabilities on the parents alpha_i
}
transformed parameters {
  vector<lower=0>[n_st_ch] alpha_ch[n_st_pr]; // mixture alpha
  vector<lower=0>[n_st_ch] alpha; // population level
  vector<lower=0>[n_st_ch] alpha_i[n_st_pr_sum];// prior for local states
  alpha = N_prior * alpha_norm;  // population-level
  for(i in 1:n_st_pr_sum){ alpha_i[i] = N_prior*alpha_i_norm[i];}// parent state-level
  // Mix alpha hyperparameter for the prior over each parent's state
  for( i_st in 1:n_st_pr ){
    alpha_ch[i_st] = rep_vector(0,n_st_ch); // initialize as zeros
    for( i in 1:n_parent ){
      // cummulative mixture contributions of parent states
      alpha_ch[i_st] += p_mix[i] * alpha_i[ sum(head(n_st_pr_i,i-1)) + i_st_pr[i_st,i] ];
    }
  }
}
model {
  alpha_norm ~ dirichlet( alpha_0 );      // prior
  N_prior    ~ student_t( 4, 1, 1 );      // prior
  p_mix      ~ dirichlet( alpha_pr_mix ); // prior
  {
    int i_st = 0;
    for( i in 1:n_parent ){
      for( j in 1:n_st_pr_i[i]){
        i_st += 1;
        alpha_i_norm[i_st] ~ dirichlet( alpha ); // prior
      }
    }
  }
  for (i_st in 1:n_st_pr){
    theta[i_st]    ~ dirichlet( alpha_ch[i_st] ); // prior
    N_ch_pr[,i_st] ~ multinomial( theta[i_st] );  // likelihood
  }
}
```

Now, we compile the **Stan** programs from file.

```r
# STAN COMPILATION ====
# Compile the Stan programs of the two model variants.
sm_hierMD    <- stan_model(file="hierMD_MLT.stan",    model_name="hierMD")
sm_hierMDmix <- stan_model(file="hierMDmix_MLT.stan", model_name="hierMDmix")
```

Estimation of the Conditional Probability Tables (CPT)

We show application of the `gen_CPT_hierMD()` function, which performs Variational Bayesian Infernce on both models to generate CPT parameters first using the `hierMD` method then the `hierMDmix` method for a single named child node. It also computes the parameters using Laplace smoothing (`BDeu`).

We could do it for every child node in the BBN, but instead, we're just going to do it on the three viewer feature nodes: `gender`, `age`, and `occupation`.

```r
# CPT ESTIMATION ====
rslt_gender <- gen_CPT_hierMD( bbn, df, ch_name = "gender" )
rslt_age    <- gen_CPT_hierMD( bbn, df, ch_name = "age" )
rslt_occ    <- gen_CPT_hierMD( bbn, df, ch_name = "occupation" )

rslt <- list(gender=rslt_gender, age = rslt_age, occupation = rslt_occ)

# # Do All Child Nodes:
# rslt <- setdiff( bbn$universe$nodes, movie ) %>%
#   set_names(.,.) %>%
#   map( ~ gen_CPT_hierMD(bbn, df, ch_name = .x) )
```

Display the CPT parameters of the `gender` node for each method.

```r
# Test the gender-node revision
theta_BDeu_eps  <- rslt_gender$theta_BDeu_eps
theta_hierMD    <- rslt_gender$theta_hierMD
theta_hierMDmix <- rslt_gender$theta_hierMDmix

df %>%
  filter(str_detect(occupation,"programmer")) %$%
  table(gender,Str.ctr_79) %>%
  {list(`OCCURRENCES: occupation == "programmer"`= .)}
```

```
## $`OCCURRENCES: occupation == "programmer"`
##       Str.ctr_79
## gender r2=-3 r3=-2 r4=-1 r5=0 r6=1 r7=2 unseen
##      F     0     0     0    0    0    0      6
##      M     0     1     0   11    3    0     48
```

```r
cpt <- bbn$cptlist$gender
n_st_pr_i <- dim(cpt)[-1]

# The 15th occupation is "programmer".
(cpt[,,15,drop=FALSE]) %>% round(3)
```

```
## , , occupation = programmer
##
##       Str.ctr_79
## gender r2=-3 r3=-2 r4=-1 r5=0  r6=1 r7=2 unseen
##      F   0.5 0.001   0.5    0 0.001  0.5  0.115
##      M   0.5 0.999   0.5    1 0.999  0.5  0.885
```

```r
theta_BDeu_eps[(15-1)*n_st_pr_i[[1]] + (1:n_st_pr_i[[1]]),] %>%
  t() %>%
  round(3) %>%
  {dimnames(.)<- dimnames(cpt)[-3]; .}
```

```
##       Str.ctr_79
## gender r2=-3 r3=-2 r4=-1 r5=0  r6=1 r7=2 unseen
##      F   0.5 0.003   0.5    0 0.001  0.5  0.111
##      M   0.5 0.997   0.5    1 0.999  0.5  0.889
```

```r
theta_hierMD[(15-1)*n_st_pr_i[[1]] + (1:n_st_pr_i[[1]]),] %>%
  t() %>%
  round(3) %>%
  {dimnames(.)<- dimnames(cpt)[-3]; .}
```

```
##       Str.ctr_79
## gender r2=-3 r3=-2 r4=-1  r5=0  r6=1  r7=2 unseen
##      F 0.341 0.193 0.348 0.054 0.107 0.294  0.108
##      M 0.659 0.807 0.652 0.946 0.893 0.706  0.892
```

```r
theta_hierMDmix[(15-1)*n_st_pr_i[[1]] + (1:n_st_pr_i[[1]]),] %>%
  t() %>%
  round(3) %>%
  {dimnames(.)<- dimnames(cpt)[-3]; .}
```

```
##       Str.ctr_79
## gender r2=-3 r3=-2 r4=-1  r5=0 r6=1  r7=2 unseen
##      F 0.432 0.205 0.281 0.053 0.15 0.405  0.125
##      M 0.568 0.795 0.719 0.947 0.85 0.595  0.875
```

Revision of the BBN

Now, let's load the CPTs into separate BBN and evaluate how they differ from the original BBN in terms of inferences.

```r
# REVISION OF BBN ====
bbn_list <- list( hierMD=bbn, hierMDmix=bbn )
for(ch_name in c("age","gender","occupation")){

  bbn_list <- revise_bbn(
    bbn_list,
    ch_name         = ch_name,
    theta_hierMD    = rslt[[ch_name]]$theta_hierMD,
    theta_hierMDmix = rslt[[ch_name]]$theta_hierMDmix
  )
}

bbn_hierMD    <- bbn_list$hierMD
bbn_hierMDmix <- bbn_list$hierMDmix
```

Impact on Bayesian Inference

The space is so high-dimensional, and we only have `N=1000` cases. So, most of the combos are not measured. Yet, the models will make inferences for any combination of node values.

(What's actually needed is feedback to the practitioner that the network is highly uncertain about any inferences in such instances. That's where having the full posterior distributions of the CPT parameters is helpful. But, we don't explore that here.)

```
# IMPACT ON BAYESIAN INFERNCE ====
# Conditional ratings distribution and expected rating given gender,
# posterior given a programmer who has seen the movie.

# First, show occurrences:
df %>%
  filter(
    str_detect(occupation,"programmer"),
    str_detect(Has_Seen,"yes"),
    str_detect(age, "yrs_33_44")
  ) %$%
  table(Str.ctr_79,gender) %>%
  {list(`OCCURRENCES: `= .)}
```

```
## $`OCCURRENCES: `
##           gender
## Str.ctr_79 F M
##     r2=-3  0 0
##     r3=-2  0 0
##     r4=-1  0 0
##     r5=0   0 3
##     r6=1   0 1
##     r7=2   0 0
##     unseen 0 0
```

```
case_profile <- list(occupation="programmer", Has_Seen = "yes", age = "yrs_33_44")
querygrain(bbn,
           nodes = c("Str.ctr_79","gender"),
           evidence = case_profile,
           type = "conditional") %T>% {print(signif(.,3))} %>%
  {list(Expected_Rating = round(t(.) %*% c(-3:2,0),2))}
```

```
##           gender
## Str.ctr_79        F        M
##     r2=-3  0.144000 0.000170
##     r3=-2  0.000327 0.000339
##     r4=-1  0.144000 0.000170
##     r5=0   0.206000 0.500000
##     r6=1   0.360000 0.499000
##     r7=2   0.144000 0.000170
##     unseen 0.000000 0.000000
```

```
## $Expected_Rating
##
## gender [,1]
##      F 0.07
##      M 0.50
```

```
querygrain(bbn_hierMD,
           nodes = c("Str.ctr_79","gender"),
           evidence = case_profile,
           type = "conditional") %T>% {print(signif(.,3))} %>%
  {list(Expected_Rating = round(t(.) %*% c(-3:2,0),2))}
```

```
##           gender
## Str.ctr_79      F       M
##     r2=-3  0.0655 0.04570
##     r3=-2  0.3700 0.06860
##     r4=-1  0.2260 0.09320
##     r5=0   0.1380 0.63200
##     r6=1   0.0943 0.00699
##     r7=2   0.1070 0.15400
##     unseen 0.0000 0.00000
```

```
## $Expected_Rating
##
## gender  [,1]
##      F -0.85
##      M -0.05
```

```
querygrain(bbn_hierMDmix,
           nodes = c("Str.ctr_79","gender"),
           evidence = case_profile,
           type = "conditional") %T>% {print(signif(.,3))} %>%
  {list(Expected_Rating = round(t(.) %*% c(-3:2,0),2))}
```

```
##           gender
## Str.ctr_79      F       M
##     r2=-3  0.0709 0.03530
##     r3=-2  0.2830 0.03750
##     r4=-1  0.2570 0.05700
##     r5=0   0.1180 0.71700
##     r6=1   0.1770 0.00291
##     r7=2   0.0948 0.15000
##     unseen 0.0000 0.00000
```

```
## $Expected_Rating
##
## gender  [,1]
##      F -0.67
##      M  0.07
```

Now, let's look at a scenario with no data: that of `occupation = "doctor"`.

```
# Conditional ratings distribution given gender,
# posterior given a doctor who has seen the movie.
# First, show occurrences:
df %>%
  filter(
    str_detect(occupation,"doctor"),
    str_detect(Has_Seen,"yes"),
    str_detect(age, "yrs_33_44")
  ) %$%
  table(Str.ctr_79,gender) %>%
  {list(`OCCURRENCES: `= .)}
```

```
## $`OCCURRENCES: `
##           gender
## Str.ctr_79 F M
##     r2=-3  0 0
##     r3=-2  0 0
```

```
##     r4=-1  0 0
##     r5=0   0 0
##     r6=1   0 0
##     r7=2   0 0
##     unseen 0 0
```

```
case_profile <- list(occupation="doctor", Has_Seen = "yes", age = "yrs_33_44")
querygrain(bbn,
           nodes = c("Str.ctr_79","gender"),
           evidence = case_profile,
           type = "conditional") %T>% {print(signif(.,3))} %>%
  {list(Expected_Rating = round(t(.) %*% c(-3:2,0),2))}
```

```
##           gender
## Str.ctr_79     F     M
##     r2=-3  0.167 0.167
##     r3=-2  0.167 0.167
##     r4=-1  0.167 0.167
##     r5=0   0.167 0.167
##     r6=1   0.167 0.167
##     r7=2   0.167 0.167
##     unseen 0.000 0.000
```

```
## $Expected_Rating
##
## gender [,1]
##      F -0.5
##      M -0.5
```

```
querygrain(bbn_hierMD,
           nodes = c("Str.ctr_79","gender"),
           evidence = case_profile,
           type = "conditional") %T>% {print(signif(.,3))} %>%
  {list(Expected_Rating = round(t(.) %*% c(-3:2,0),2))}
```

```
##           gender
## Str.ctr_79      F       M
##     r2=-3  0.0821 0.05060
##     r3=-2  0.0117 0.36100
##     r4=-1  0.4260 0.39600
##     r5=0   0.2300 0.09510
##     r6=1   0.2400 0.08940
##     r7=2   0.0105 0.00825
##     unseen 0.0000 0.00000
```

```
## $Expected_Rating
##
## gender  [,1]
##      F -0.43
##      M -1.16
```

```
querygrain(bbn_hierMDmix,
           nodes = c("Str.ctr_79","gender"),
           evidence = case_profile,
           type = "conditional") %T>% {print(signif(.,3))} %>%
  {list(Expected_Rating = round(t(.) %*% c(-3:2,0),2))}
```

```
##           gender
## Str.ctr_79       F       M
##     r2=-3  0.11700 0.04000
##     r3=-2  0.00857 0.29800
##     r4=-1  0.34300 0.52800
##     r5=0   0.25700 0.08700
##     r6=1   0.26900 0.03790
##     r7=2   0.00509 0.00914
##     unseen 0.00000 0.00000
```

```
## $Expected_Rating
##
## gender  [,1]
##      F -0.43
##      M -1.19
```

Impact on Generalized Bayes Factor, GBF(H:E)

The Generalized Bayes Factor, GBF(H:E), has been shown to be a good metric for hypothesis (H) confirmation given evidence (E) and for use in generating relevant explanations (a ranked list of H's) of observed evidence (E). However, it is very sensitive to noisy estimates of prior and posterior probabilities for sparsely measured cases.

Given that the `hierMD` and `hierMDmix` approaches both spread probability mass throughout the sparse CPTs that is more consistent with the population-level distributions, we would expect the GBF(H:E) for cases of either sparsely measured hypotheses H or evidence E to be less noisy, though somewhat biased due to the shrinkage towards the population marginal distributions that these priors induce.

```
# IMPACT ON GENERALIZED BAYES FACTOR ====
# Compare models impact on GBF(H:E) under different case profiles as
# the "hypothesis" H, given the "evidence" E = LIKE = "yes".
list( BDeu = bbn, hierMD = bbn_hierMD, hierMDmix = bbn_hierMDmix ) %>%
  imap_dfr(
    ~ gbf(bbn = .x, gndr = "F", occ = "doctor", age = "yrs_33_44") %$%
      map(at_cond,signif,3) %>%
      as_tibble() %>%
      mutate(model=.y) %>%
      select(model,everything())
  )
```

```
## # A tibble: 3 x 7
##   model       gbf     wte     joint joint_post prior_odds post_odds
##   <chr>     <dbl>   <dbl>     <dbl>      <dbl>      <dbl>     <dbl>
## 1 BDeu      0.959 -0.18   0.0000346  0.0000332  0.0000346 0.0000332
## 2 hierMD    0.868 -0.612  0.00369    0.0032     0.0037    0.00321
## 3 hierMDmix 0.978 -0.0979 0.00364    0.00356    0.00366   0.00357
```

```
list( BDeu = bbn, hierMD = bbn_hierMD, hierMDmix = bbn_hierMDmix ) %>%
  imap_dfr(
    ~ gbf(bbn = .x, gndr = "M", occ = "programmer", age = "yrs_33_44") %$%
      map(at_cond,signif,3) %>%
      as_tibble() %>%
      mutate(model=.y) %>%
      select(model,everything())
  )
```

```
## # A tibble: 3 x 7
```

```
##   model       gbf   wte   joint joint_post prior_odds post_odds
##   <chr>     <dbl> <dbl>   <dbl>      <dbl>      <dbl>     <dbl>
## 1 BDeu       1.98  2.97 0.034       0.0651    0.0352     0.0697
## 2 hierMD     1.44  1.58 0.00795     0.0114    0.00801    0.0115
## 3 hierMDmix  1.61  2.07 0.0101      0.0161    0.0102     0.0164
```

```
list( BDeu = bbn, hierMD = bbn_hierMD, hierMDmix = bbn_hierMDmix ) %>%
  imap_dfr(
    ~ gbf(bbn = .x, gndr = "F", occ = "programmer", age = "yrs_33_44") %$%
      map(at_cond,signif,3) %>%
      as_tibble() %>%
      mutate(model=.y) %>%
      select(model,everything())
  )
```

```
## # A tibble: 3 x 7
##   model       gbf   wte    joint joint_post prior_odds post_odds
##   <chr>     <dbl> <dbl>    <dbl>      <dbl>      <dbl>     <dbl>
## 1 BDeu      1.36   1.35 0.00004   0.0000545   0.00004  0.0000546
## 2 hierMD    0.613 -2.12 0.00069   0.000423    0.000691 0.000424
## 3 hierMDmix 0.717 -1.45 0.000407  0.000292    0.000408 0.000292
```

```
list( BDeu = bbn, hierMD = bbn_hierMD, hierMDmix = bbn_hierMDmix ) %>%
  imap_dfr(
    ~ gbf(bbn= .x, gndr = "F", occ = "administrator", age = "yrs_33_44") %$%
      map(at_cond,signif,3) %>%
      as_tibble() %>%
      mutate(model=.y) %>%
      select(model,everything())
  )
```

```
## # A tibble: 3 x 7
##   model       gbf    wte     joint joint_post prior_odds post_odds
##   <chr>     <dbl>  <dbl>     <dbl>      <dbl>      <dbl>     <dbl>
## 1 BDeu      0.841 -0.754 0.0000614  0.0000516  0.0000614 0.0000516
## 2 hierMD    1.12   0.496 0.00237    0.00266    0.00238   0.00267
## 3 hierMDmix 1.27   1.05  0.00213    0.00271    0.00214   0.00272
```

Thoughts

Of course, much more work would be needed to show if and when the `hierMDmix` method adds value over that of the `hierMD` method. But, *ACZ* have already shown the value of `hierMD` over traditional smoothing calculation of Bayesian network CPTs.

It is nice to see that either of these approaches is so easily implemented using **Stan**. Moreover, *ACZ* provide **R** source code – "Hierarchical BN parameter estimation" – to perform the variational Bayesian inference for `hierMD` both with and without using **Stan**.

Finally, having the full posterior (approximately) of the CPT parameters is a nice feature of these two methods. In the future, we should exploit this by reporting or visualizing the uncertainty quantification in risk assessment and decision analysis.

About

-Michael L. Thompson,

LinkedIn profile