VFQ 25 - Example

Installing smartPROM

If you are using `smartPROM` for the first time, install it from the github repo. We need the `devtools` library to install libraries from github.

```
library(devtools)
install_github("byet/smartPROM")
```

Analysis

```
library(smartPROM)
```

```
##
## Attaching package: 'smartPROM'
```

```
## The following object is masked from 'package:base':
##
##     cat
```

Read the dataset

```
df <- read.csv("./data/Cleaned_and_scored_VFQ25.csv")
```

Discretize the states

In this example, we will discretize the vfq overall score into two states: `H` = above 75, `L` = below 75.

> You can discretize into more than two intervals by providing a vector to cutoff. Note that, number of states should be one more than number of cutoffs. e.g. 'discretize(df$OverallScore, cutoff=C(25,75), states = c('L','M','H'))

```
intv <- discretize(df$OverallScore, cutoff=75, states = c('L','H'))
```

We also have a function that ensures disretised intervals are larger than MCID and have about the same size. But this'd be an overkill for this example.

```
intv <- mipm_intervals(df$OverallScore, threshold=75, MCID= 15, states = c('L','H'), n_interval = 2, Lo
```

Measured and Target variables.

We will use questions 1-14, 18-25 as our measured variables. These will be potential inputs that will be asked in our evaluations. We ignored the questions about driving in this example as they had many missing values (non-drivers).

Our target variable is the discretized score we created above. We name it `lf` in our analysis.

```
measured <- paste0("vfq", c(1,2,3,4,5,6,7,8,9,10,11,12,13,14,18,19,20,21,22,23,24,25))
target <- "lf"
n_measured <- length(measured)

# Dataset composed of just inputs and target variable.
dfq <- df[measured]
dfq[target] <- intv
```

Learn BN

We use a Naive BN structure in this example. We could have used BN structure built by experts or data-driven structure learning algorithms.

```
naive_bn_dfq <- naivebn_arcs(target,measured)
bn <- learn_bn(dfq, arc_set= naive_bn_dfq, na_omit = TRUE)
```

Computerized Adaptive Testing with BNs

`cat_iterations` iteratively asks questions from `measured` by using the CAT algorithm. For, every question selected by CAT, we enter the data regarding this question to the BN and get the posterior probability of `target` after entering the data.

Below, we run `cat_iterations` for the first row of our dataset.

```
res_iter <- cat_iterations(bn, data = dfq[1,], target=target, query = measured, num_iter = n_measured ,
```

Results

`distance_to_all_queries` compares the probability distribution of target variables when we enter the responses of first x questions entered with CAT to when we enter the responses for all the questions.

```
avedist <- distance_to_all_queries(res_iter)
print(avedist)
```

```
##            lf            lf            lf            lf            lf
##  1.014484e-02  2.478380e-03  7.252465e-04  2.891743e-05  8.610211e-06
##            lf            lf            lf            lf            lf
##  2.934647e-06  5.769667e-07  1.438398e-07  1.585806e-08  3.477396e-09
##            lf            lf            lf            lf            lf
##  3.081990e-10  9.663592e-12  3.323775e-12  4.292030e-13  1.509818e-13
##            lf            lf            lf            lf            lf
##  7.681942e-14  2.214861e-15  2.214861e-15  2.184704e-16 -1.754270e-18
##            lf            lf
##  0.000000e+00  0.000000e+00
```

Below we use `errorAllQuestion` to compare the accuracy of predicting target variables when different number of inputs are entered with CAT. This is for only the first row of data.

```
errorAllQuestions(res_iter$bns, dfq[1,], measured,numericStates = FALSE)
```

```
##               [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## Accuracy 0.8181818 0.8181818 0.8181818 0.8181818 0.8181818 0.8181818 0.8181818
## MAE             NA        NA        NA        NA        NA        NA        NA
##               [,8]      [,9]     [,10]     [,11]     [,12]     [,13]     [,14]
## Accuracy 0.8181818 0.8181818 0.8181818 0.8181818 0.8181818 0.8181818 0.8181818
## MAE             NA        NA        NA        NA        NA        NA        NA
##              [,15]     [,16]     [,17]     [,18]     [,19]     [,20]     [,21]
## Accuracy 0.8181818 0.8181818 0.8181818 0.8181818 0.8181818 0.8636364 0.9090909
## MAE             NA        NA        NA        NA        NA        NA        NA
##              [,22]
## Accuracy 0.9090909
## MAE             NA
```

Below we use `errorAllQuestion` to compare the accuracy of predicting responses to unanswered questions when different number of inputs are entered with CAT. This is for only the first row of data.

```
errorAllQuestions(res_iter$bns, dfq[1,], target, numericStates = FALSE)
```

```
##          [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13]
## Accuracy    1    1    1    1    1    1    1    1    1     1     1     1     1
## MAE        NA   NA   NA   NA   NA   NA   NA   NA   NA    NA    NA    NA    NA
##          [,14] [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22]
## Accuracy     1     1     1     1     1     1     1     1     1
## MAE         NA    NA    NA    NA    NA    NA    NA    NA    NA
```

To get the error for all data, we need to compute this for all rows of data and get the average.

Running cat for all the data

Below we run CAT for all rows of data.

```
pb <- txtProgressBar(min = 0, max = nrow(dfq), style = 3)
```

```
##   |                                                                      |
```

```
res <- lapply(1:nrow(dfq),
              function(x, data){
                setTxtProgressBar(pb, x)
                cat_iterations(bn = bn, target = target, query = measured, data = data[x,], num_iter=n_
              dfq)
```

```
##   |                                                                      |
```

Below we compute the area under the curve for target variable prediction when different number of questions are selected with CAT.

```
perf <- performance(res, dfq, targetVars = target)
perf
```

```
##           lf
## 1  0.9029526
## 2  0.9358327
## 3  0.9447682
## 4  0.9759000
## 5  0.9761461
## 6  0.9782699
## 7  0.9834758
## 8  0.9843564
## 9  0.9868816
## 10 0.9867392
## 11 0.9869593
## 12 0.9881896
## 13 0.9883061
## 14 0.9875939
## 15 0.9871018
## 16 0.9870500
## 17 0.9873608
## 18 0.9879565
## 19 0.9881637
## 20 0.9881896
## 21 0.9873867
## 22 0.9874385
```