

CopReg Vignette: Donkey Dataset

24 April 2022

1. Data loading and univariate margins

Setup and load xdonkey dataset

```
library(CopReg)
library(sn)
data("xdonkey")
head(xdonkey)
#>   bcs age sex slength sgirth sheight sweight
#> 1   5  1  3   0.78  0.90   0.90   0.77
#> 2   4  1  3   0.91  0.97   0.94   1.00
#> 3   3  1  3   0.74  0.93   0.95   0.74
#> 4   5  1  1   0.87  1.09   0.96   1.16
#> 5   4  1  1   0.79  0.98   0.91   0.91
#> 6   3  1  1   0.86  1.02   0.98   1.05
```

Train/holdout split

```
set.seed(1234)
ntot <- nrow(xdonkey)
itrain <- sample(ntot, 400)
# training set
donkey.tr <- data.frame(xdonkey[itrain, ])
head(donkey.tr)
#>   bcs age sex slength sgirth sheight sweight
#> 284  7  4  2   1.01  1.34   1.10   2.10
#> 101  6  2  1   1.07  1.20   1.00   1.65
#> 400  6  4  1   0.99  1.25   1.03   1.80
#> 98   4  3  3   0.94  1.08   0.96   1.31
#> 103  5  6  1   0.96  1.12   0.99   1.37
#> 326  4  2  1   0.86  1.12   1.02   1.30
summary(donkey.tr)
#>   bcs          age          sex          slength
#> Min.   :3.000   Min.   :1.00   Min.   :1.000   Min.   :0.7200
#> 1st Qu.:4.000   1st Qu.:2.00   1st Qu.:1.000   1st Qu.:0.9200
#> Median :5.000   Median :4.00   Median :2.000   Median :0.9650
#> Mean   :4.787   Mean   :3.43   Mean   :1.917   Mean   :0.9595
#> 3rd Qu.:5.000   3rd Qu.:4.00   3rd Qu.:3.000   3rd Qu.:1.0100
#> Max.   :7.000   Max.   :6.00   Max.   :3.000   Max.   :1.1100
#>   sgirth      sheight      sweight
#> Min.   :0.930   Min.   :0.890   Min.   :0.740
#> 1st Qu.:1.127   1st Qu.:0.990   1st Qu.:1.390
#> Median :1.170   Median :1.020   Median :1.550
#> Mean   :1.163   Mean   :1.016   Mean   :1.529
```

```

#> 3rd Qu.:1.210 3rd Qu.:1.040 3rd Qu.:1.700
#> Max. :1.340 Max. :1.160 Max. :2.300

# holdout set
donkey.ho <- data.frame(xdonkey[-itrain, ])
head(donkey.ho)
#>  bcs age sex slength sgirth sheight sweight
#> 1  5  1  3  0.78  0.90  0.90  0.77
#> 2  4  1  3  0.91  0.97  0.94  1.00
#> 5  4  1  1  0.79  0.98  0.91  0.91
#> 7  4  1  3  0.83  1.06  0.96  1.08
#> 8  3  1  3  0.77  0.95  0.89  0.86
#> 9  5  1  3  0.46  0.66  0.71  0.27
summary(donkey.ho)
#>      bcs      age      sex      slength
#> Min. :3.000 Min. :1.000 Min. :1.000 Min. :0.4600
#> 1st Qu.:4.000 1st Qu.:2.000 1st Qu.:1.000 1st Qu.:0.9100
#> Median :5.000 Median :4.000 Median :2.000 Median :0.9700
#> Mean :4.799 Mean :3.458 Mean :1.972 Mean :0.9492
#> 3rd Qu.:5.000 3rd Qu.:4.000 3rd Qu.:3.000 3rd Qu.:1.0000
#> Max. :7.000 Max. :6.000 Max. :3.000 Max. :1.1200
#>      sgirth      sheight      sweight
#> Min. :0.660 Min. :0.710 Min. :0.270
#> 1st Qu.:1.127 1st Qu.:0.990 1st Qu.:1.377
#> Median :1.170 Median :1.010 Median :1.560
#> Mean :1.151 Mean :1.007 Mean :1.498
#> 3rd Qu.:1.200 3rd Qu.:1.040 3rd Qu.:1.700
#> Max. :1.310 Max. :1.100 Max. :2.040
# plots will reveal one outlier in the holdout set
ii <- which(donkey.ho$sweight==min(donkey.ho$sweight)); print(ii)
#> [1] 6
print(donkey.ho[ii,])
#>  bcs age sex slength sgirth sheight sweight
#> 9  5  1  3  0.46  0.66  0.71  0.27
# remove the outlier so that training set has 399 observations
donkey.ho <- donkey.ho[-ii,]

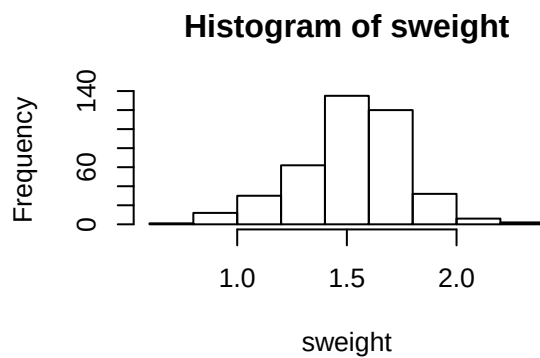
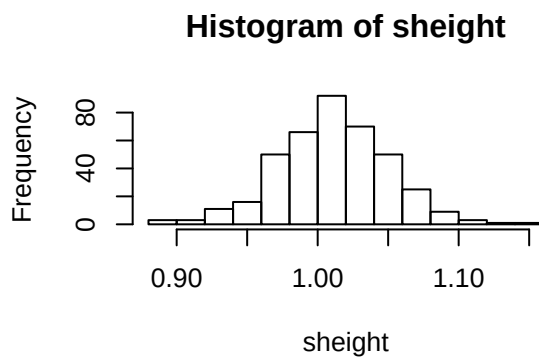
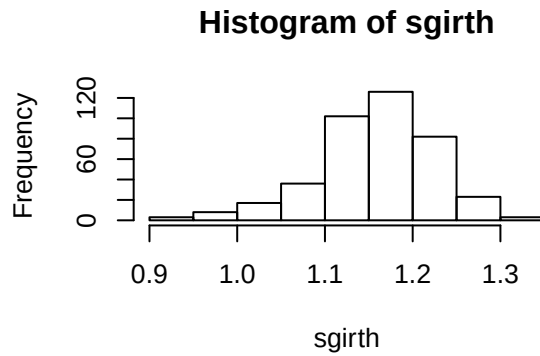
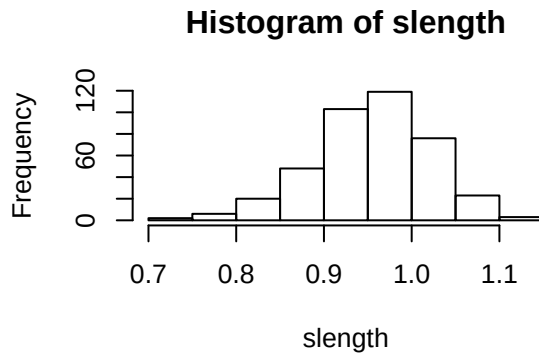
```

Exploratory data analysis

```

attach(donkey.tr)
par(mfrow=c(2,2))
hist(slength) # left skewed
hist(sgirth) # left skewed
hist(sheight) # left skewed?
hist(sweight) # left skewed

```



Fit marginal distributions

```
# fit skew-normal distributions
nn <- nrow(donkey.tr)
pvec <- ((1:nn) - .5) / nn
```

One way to use skew-normal with the original parameters

```
# pdf
dskewn <- function(y,param)
{ dsn(y,xi=param[1],omega=param[2],alpha=param[3]) }

# cdf
pskewn <- function(y,param)
{ psn(y,xi=param[1],omega=param[2],alpha=param[3]) }

# log of skew-normal density
logdskewn <- function(y,param)
{ dsn(y,xi=param[1],omega=param[2],alpha=param[3],log=T) }

# inverse cdf
qskewn <- function(p,param)
{ qst(p,xi=param[1],omega=param[2],alpha=param[3]) }

# negative log-likelihood for skew-normal density
```

```

sknnllk <- function(param,xdat)
{ if(param[2]<=0) return(1.e10)
  nllk <- -logdskewn(xdat,param)
  sum(nllk)
}

```

slength

```

model_slength <- selm(slength ~ 1)
summary(model_slength)
#> Call: selm(formula = slength ~ 1)
#> Number of observations: 400
#> Family: SN
#> Estimation method: MLE
#> Log-likelihood: 522.3466
#> Parameter type: CP
#>
#> CP residuals:
#>      Min      1Q   Median      3Q      Max
#> -0.239449 -0.039449  0.005551  0.050551  0.150551
#>
#> Regression coefficients
#>      estimate  std.err  z-ratio Pr{>|z|}
#> mean 9.594e-01 3.327e-03 2.884e+02      0
#>
#> Parameters of the SEC random component
#>      estimate std.err
#> s.d.    0.06661  0.003
#> gamma1 -0.43337  0.113
# parameters are mean, s.d. and gamma, not location, scale and skew as
#   in the papers of Azzalini et al.
u_slength <- psn(slength, dp = coef(model_slength, "DP"))

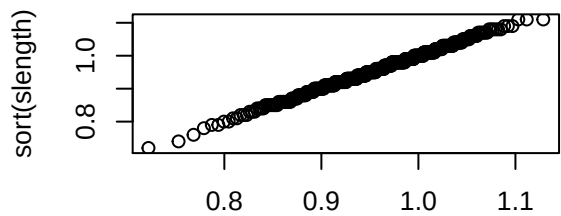
# alternative for comparison
startl <- c(mean(slength),sd(slength),-.3)
mll <- nlm(sknnllk,p=startl,hessian=T,print.level=1,xdat=slength)
#> iteration = 0
#> Step:
#> [1] 0 0 0
#> Parameter:
#> [1] 0.95945000 0.06669196 -0.30000000
#> Function Value
#> [1] -504.7831
#> Gradient:
#> [1] -1453.5735 -317.0826 -73.8443
#>
#> iteration = 28
#> Parameter:
#> [1] 1.02627162 0.09434959 -1.92756346
#> Function Value
#> [1] -522.3466
#> Gradient:
#> [1] 2.083480e-04 -1.334541e-04 2.124739e-06

```

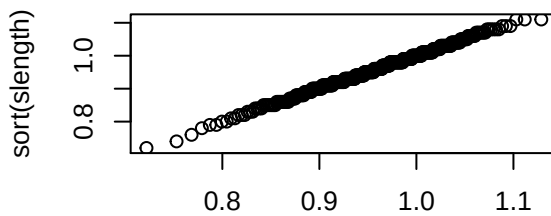
```

#>
#> Relative gradient close to zero.
#> Current iterate is probably solution.
parl <- mll$estimate
ulen <- pskewn(slength,parl)
summary(ulen)
#>   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#> 0.00117 0.25905 0.50198 0.49975 0.77023 0.99586
par(mfrow=c(2,2))
plot(qsn(pvec, dp = coef(model_slength, "DP")), sort(slength))
plot(qskewn(pvec,parl),sort(slength))
plot(ulen,u_slength) # same (2 different parametrizations of skew-normal)

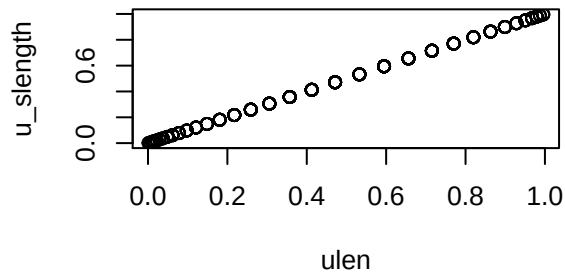
```



qsn(pvec, dp = coef(model_slength, "DP"))



qskewn(pvec, parl)



sgirth

```

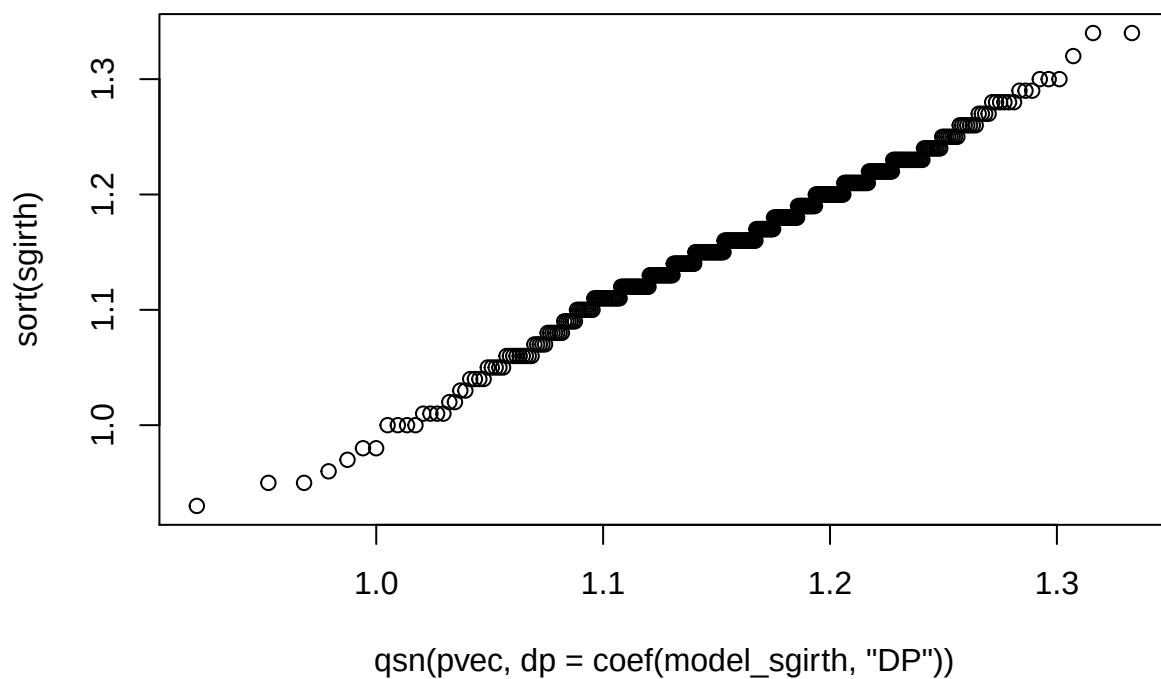
model_sgirth <- selm(sgirth ~ 1)
summary(model_sgirth)
#> Call: selm(formula = sgirth ~ 1)
#> Number of observations: 400
#> Family: SN
#> Estimation method: MLE
#> Log-likelihood: 517.5439
#> Parameter type: CP
#>
#> CP residuals:

```

```

#>      Min      1Q   Median      3Q      Max
#> -0.232293 -0.034793  0.007707  0.047707  0.177707
#>
#> Regression coefficients
#>      estimate std.err  z-ratio Pr{>|z|}
#> mean 1.162e+00 3.368e-03 3.451e+02      0
#>
#> Parameters of the SEC random component
#>      estimate std.err
#> s.d.    0.06748    0.003
#> gamma1 -0.44188    0.095
plot(qsn(pvec, dp = coef(model_sgirth, "DP")), sort(sgirth))

```



```
u_sgirth <- psn(sgirth, dp = coef(model_sgirth, "DP"))
```

sheight

```

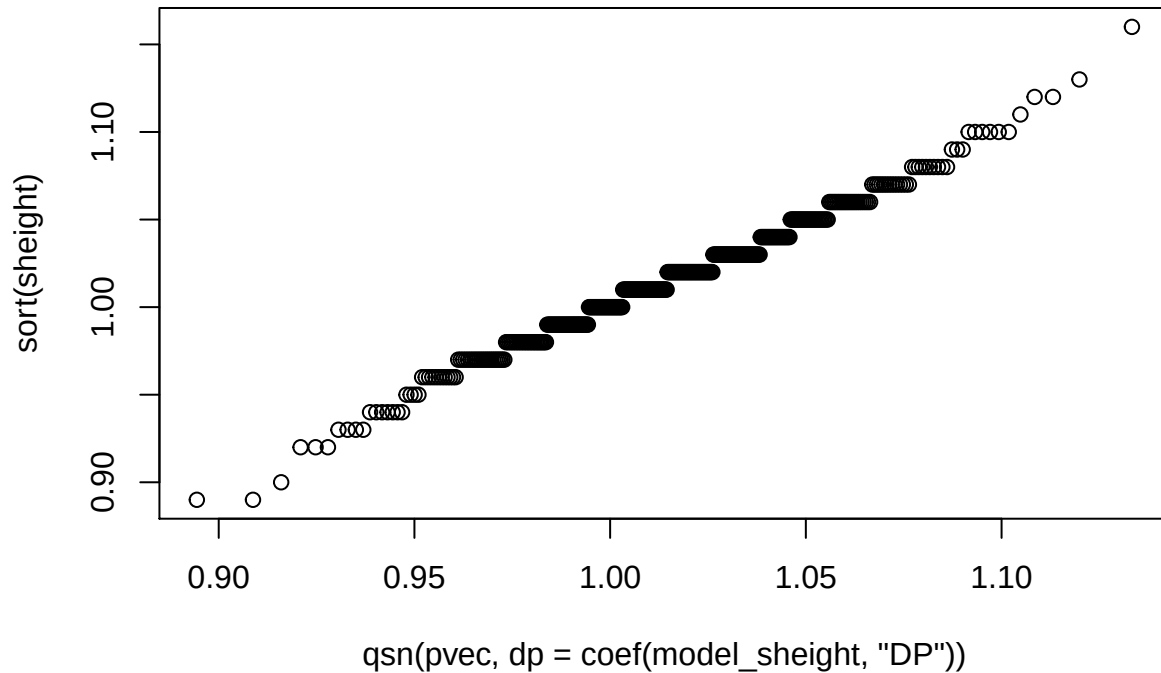
model_sheight <- selm(sheight ~ 1)
summary(model_sheight)
#> Call: selm(formula = sheight ~ 1)
#> Number of observations: 400
#> Family: SN
#> Estimation method: MLE
#> Log-likelihood: 725.8052
#> Parameter type: CP
#>

```

```

#> CP residuals:
#>      Min      1Q   Median      3Q      Max
#> -0.125796 -0.025796  0.004204  0.024204  0.144204
#>
#> Regression coefficients
#>      estimate std.err  z-ratio Pr{>|z|}
#> mean 1.016e+00 1.971e-03 5.153e+02      0
#>
#> Parameters of the SEC random component
#>      estimate std.err
#> s.d.    0.03942    0.001
#> gamma1 -0.03657    0.113
plot(qsn(pvec, dp = coef(model_sheight, "DP")), sort(sheight))

```



```
u_sheight <- psn(sheight, dp = coef(model_sheight, "DP"))
```

sweight

```

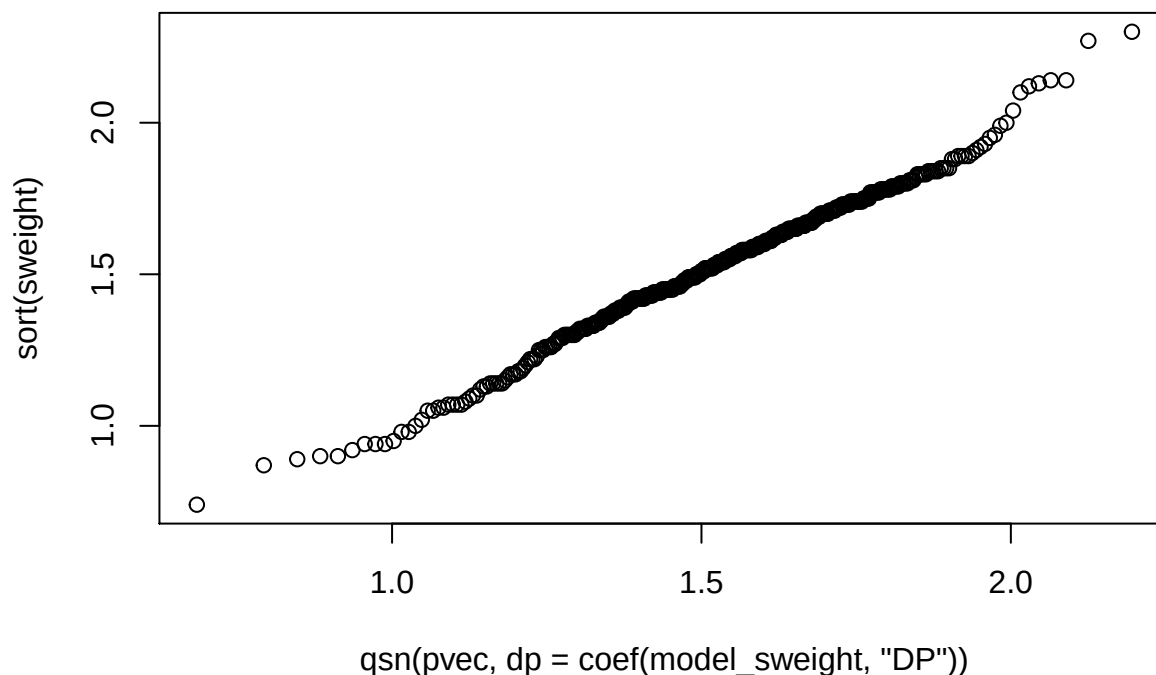
model_sweight <- selm(sweight ~ 1)
summary(model_sweight)
#> Call: selm(formula = sweight ~ 1)
#> Number of observations: 400
#> Family: SN
#> Estimation method: MLE
#> Log-likelihood: -5.316139
#> Parameter type: CP

```

```

#>
#> CP residuals:
#>      Min      1Q   Median      3Q      Max
#> -0.78904 -0.13904  0.02096  0.17096  0.77096
#>
#> Regression coefficients
#>      estimate std.err  z-ratio Pr{>|z|}
#> mean   1.52904   0.01233 124.00930      0
#>
#> Parameters of the SEC random component
#>      estimate std.err
#> s.d.    0.2468   0.009
#> gamma1  -0.2949   0.108
plot(qsn(pvec, dp = coef(model_sweight, "DP")), sort(sweight))

```



```
u_sweight <- psn(sweight, dp = coef(model_sweight, "DP"))
```

Ordinal variables

```

# convert ordinal variables to cdf scale
bcs_ecdf <- ecdf(bcs)
age_ecdf <- ecdf(age)
sex_ecdf <- ecdf(sex)
udata <-
  cbind(bcs_ecdf(bcs), age_ecdf(age), sex_ecdf(sex),
        u_slength, u_sgirth, u_sheight, u_sweight)

```

```

colnames(udata) <-
  c("ubcs", "uage", "usex", "ulen", "ugir", "uhei", "uwei")
head(udata)
#>      ubcs   uage   usex      ulen      ugir      uhei      uwei
#> [1,] 1.0000 0.8425 0.6200 0.77023096 0.9992273 0.98448883 0.9946082
#> [2,] 0.9725 0.2825 0.4625 0.96745160 0.6954898 0.34227732 0.6758460
#> [3,] 0.9725 0.8425 0.4625 0.65616212 0.9178217 0.63884112 0.8692845
#> [4,] 0.3125 0.4325 1.0000 0.35696273 0.1167062 0.07930054 0.1829802
#> [5,] 0.8675 1.0000 0.4625 0.47136830 0.2487870 0.25511552 0.2484735
#> [6,] 0.3125 0.2825 0.4625 0.07801311 0.2487870 0.54010496 0.1733619
summary(udata)
#>      ubcs      uage      usex      ulen
#> Min.   :0.0600   Min.   :0.0850   Min.   :0.4625   Min.   :0.00117
#> 1st Qu.:0.3125   1st Qu.:0.2825   1st Qu.:0.4625   1st Qu.:0.25905
#> Median :0.8675   Median :0.8425   Median :0.6200   Median :0.50198
#> Mean   :0.6936   Mean   :0.6247   Mean   :0.6916   Mean   :0.49975
#> 3rd Qu.:0.8675   3rd Qu.:0.8425   3rd Qu.:1.0000   3rd Qu.:0.77023
#> Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :0.99586
#>      ugir      uhei      uwei
#> Min.   :0.001732   Min.   :0.0008725   Min.   :0.002224
#> 1st Qu.:0.282344   1st Qu.:0.2551155   1st Qu.:0.273245
#> Median :0.513692   Median :0.5401050   Median :0.513408
#> Mean   :0.503033   Mean   :0.4993627   Mean   :0.502526
#> 3rd Qu.:0.751506   3rd Qu.:0.7293246   3rd Qu.:0.749913
#> Max.   :0.999227   Max.   :0.9999007   Max.   :0.999805
# clip the values to [1e-5, 1-1e-5]
udata <- pmax(pmin(udata, 1-1e-5), 1e-5)
detach(donkey.tr)
rm(pvec, u_slength, u_sgirth, u_sheight, u_sweight)

```

2. Get correlation matrix

```

rtrain <- vWcor(udata, is_disc=c(T, T, T, F, F, F, F))
print(round(rtrain,4))
#>      [,1] [,2] [,3] [,4] [,5] [,6] [,7]
#> [1,] 1.0000 0.1934 -0.0968 0.4077 0.4986 0.2561 0.5582
#> [2,] 0.1934 1.0000 -0.0245 0.5089 0.5971 0.3807 0.6106
#> [3,] -0.0968 -0.0245 1.0000 -0.1473 -0.1440 0.0226 -0.0955
#> [4,] 0.4077 0.5089 -0.1473 1.0000 0.6222 0.5322 0.7633
#> [5,] 0.4986 0.5971 -0.1440 0.6222 1.0000 0.6574 0.8750
#> [6,] 0.2561 0.3807 0.0226 0.5322 0.6574 1.0000 0.6736
#> [7,] 0.5582 0.6106 -0.0955 0.7633 0.8750 0.6736 1.0000

```

3. Find vine structure based on different optimization criteria

```

vine_objects <- list()

vine_objects[["MSTx"]] <-
  getVinearray(rtrain, nn, method = "MSTx", cfi_bd = 1.01)
print(vine_objects[["MSTx"]])
#> $vine_array

```

```

#>      [,1] [,2] [,3] [,4] [,5] [,6] [,7]
#> [1,]    1    1    5    5    4    5    5
#> [2,]    0    5    1    2    5    4    4
#> [3,]    0    0    2    1    2    3    6
#> [4,]    0    0    0    4    1    2    3
#> [5,]    0    0    0    0    3    1    2
#> [6,]    0    0    0    0    0    6    1
#> [7,]    0    0    0    0    0    0    7
#>
#> $trunc_level
#> [1] 6
#>
#> $cfi
#> [1] 1
#>
#> $cfi_vector
#> [1] 0.8064668 0.9345314 0.9596194 0.9622187 0.9720750 1.0000000
#>
#> $treeweight
#> [1] -3.25477382 -0.52696675 -0.11078460 -0.01794200 -0.04459521 -0.11468165
#>
#> $fitval
#> [1] 0.7997490 0.9292330 0.9564545 0.9608632 0.9718209 1.0000000
#>
#> $pcmat
#>      [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
#> [1,]    0 0.4985644 0.5971437 0.6221503 -0.1473335070 0.65744548 0.8749612
#> [2,]    0 0.0000000 -0.1500763 0.2187514 -0.0675284889 0.20882165 0.5776560
#> [3,]    0 0.0000000 0.0000000 0.1829752 0.0963725496 0.17717346 0.1867718
#> [4,]    0 0.0000000 0.0000000 0.0000000 -0.0005886417 -0.08727958 0.1012026
#> [5,]    0 0.0000000 0.0000000 0.0000000 0.0000000000 -0.16327461 0.1319897
#> [6,]    0 0.0000000 0.0000000 0.0000000 0.0000000000 0.00000000 0.3291657

vine_objects[["MSTleaf"]] <-
  getVinearray(rtrain, nn, method = "MSTleaf", cfi_bd = 1.01)
print(vine_objects[["MSTleaf"]])
#> $vine_array
#>      [,1] [,2] [,3] [,4] [,5] [,6] [,7]
#> [1,]    1    1    5    5    4    5    5
#> [2,]    0    5    1    2    5    4    4
#> [3,]    0    0    2    1    2    3    6
#> [4,]    0    0    0    4    1    2    3
#> [5,]    0    0    0    0    3    1    2
#> [6,]    0    0    0    0    0    6    1
#> [7,]    0    0    0    0    0    0    7
#>
#> $trunc_level
#> [1] 6
#>
#> $cfi
#> [1] 1
#>
#> $cfi_vector

```

```

#> [1] 0.8064668 0.9345314 0.9596194 0.9622187 0.9720750 1.0000000
#>
#> $treeweight
#> [1] -3.25477382 -0.52696675 -0.11078460 -0.01794200 -0.04459521 -0.11468165
#>
#> $fitval
#> [1] 0.7997490 0.9292330 0.9564545 0.9608632 0.9718209 1.0000000
#>
#> $pcmat
#>      [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
#> [1,]  NA 0.4985644 0.5971437 0.6221503 -0.1473335070 0.65744548 0.8749612
#> [2,]  NA      NA -0.1500763 0.2187514 -0.0675284889 0.20882165 0.5776560
#> [3,]  NA      NA      NA 0.1829752 0.0963725496 0.17717346 0.1867718
#> [4,]  NA      NA      NA      NA -0.0005886417 -0.08727958 0.1012026
#> [5,]  NA      NA      NA      NA      NA -0.16327461 0.1319897
#> [6,]  NA      NA      NA      NA      NA      NA 0.3291657
#> [7,]  NA      NA      NA      NA      NA      NA      NA

vine_objects[["forwardD"]] <-
  getVinearray(rtrain, nn, method = "forwardD", cfi_bd = 1.01)
print(vine_objects[["forwardD"]])
#> $vine_array
#>      [,1] [,2] [,3] [,4] [,5] [,6] [,7]
#> [1,]    5    5    4    1    6    2    5
#> [2,]    0    4    5    4    1    6    4
#> [3,]    0    0    1    5    4    1    1
#> [4,]    0    0    0    6    5    4    6
#> [5,]    0    0    0    0    2    5    2
#> [6,]    0    0    0    0    0    3    3
#> [7,]    0    0    0    0    0    0    7
#>
#> $trunc_level
#> [1] 6
#>
#> $cfi
#> [1] 1
#>
#> $cfi_vector
#> [1] 0.5804598 0.7793253 0.9088323 0.9808250 0.9989573 1.0000000
#>
#> $treeweight
#> [1] -2.34684869 -0.81139107 -0.53026121 -0.29671254 -0.07784155 -0.00668897
#>
#> $fitval
#> [1] 0.5766576 0.7760291 0.9063226 0.9792295 0.9983564 1.0000000
#>
#> $pcmat
#>      [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
#> [1,]    0 0.6221503 0.4076997 0.2560517 0.3807339 -0.02447183 0.87496122
#> [2,]    0 0.0000000 0.3425961 0.4846908 0.1072588 0.03451045 0.57765604
#> [3,]    0 0.0000000 0.0000000 0.5062722 0.3791564 -0.10295401 0.25688473
#> [4,]    0 0.0000000 0.0000000 0.0000000 0.4373555 -0.16628895 0.23411407
#> [5,]    0 0.0000000 0.0000000 0.0000000 0.0000000 -0.17501500 0.21367556

```

```

#> [6,] 0 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.08164955

vine_objects[["forwardR"]] <-
  getVinearray(rtrain, nn, method = "forwardR", cfi_bd = 1.01)
print(vine_objects[["forwardR"]])
#> $vine_array
#>      [,1] [,2] [,3] [,4] [,5] [,6] [,7]
#> [1,] 5 5 5 5 5 4 5
#> [2,] 0 4 4 4 4 5 4
#> [3,] 0 0 1 1 1 6 1
#> [4,] 0 0 0 6 6 1 6
#> [5,] 0 0 0 0 2 2 2
#> [6,] 0 0 0 0 0 3 3
#> [7,] 0 0 0 0 0 0 7
#>
#> $trunc_level
#> [1] 6
#>
#> $cfi
#> [1] 1
#>
#> $cfi_vector
#> [1] 0.8064668 0.9340542 0.9707023 0.9852988 0.9989573 1.0000000
#>
#> $treeweight
#> [1] -3.25477382 -0.52504989 -0.15722420 -0.06613797 -0.05986919 -0.00668897
#>
#> $fitval
#> [1] 0.7997490 0.9287620 0.9673945 0.9836456 0.9983564 1.0000000
#>
#> $pcmat
#>      [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
#> [1,] 0 0.6221503 0.4985644 0.6574455 0.59714371 -0.147333507 0.87496122
#> [2,] 0 0.0000000 0.1436921 0.2088216 0.21875142 -0.067528489 0.57765604
#> [3,] 0 0.0000000 0.0000000 -0.1444704 -0.18796494 0.177173458 0.25688473
#> [4,] 0 0.0000000 0.0000000 0.0000000 -0.09834595 0.007091612 0.23411407
#> [5,] 0 0.0000000 0.0000000 0.0000000 0.0000000 0.114240494 0.21367556
#> [6,] 0 0.0000000 0.0000000 0.0000000 0.0000000 0.000000000 0.08164955
# Note that only "forwardD" is different from the other three
# The vine array for "forwardR" is different from "MSTx" but the vine is the same

```

4. Fit vine copula prediction models

Same vine arrays for MSTx and MSTleaf

copula regression using vine arrays from forwardD, forwardR and MSTx for sensitivity analysis

forwardD

```

fit_forwardD <- CopReg( data = udata,
  vine_array = vine_objects[["forwardD"]]$vine_array,
  family_set = c(1,2,4,5,7,10,14,17,20),

```

```

is_disc = c(T, T, T, F, F, F, F) )
names(fit_forwardD)
#> [1] "vine_array_1tod"      "vine_array"          "family"
#> [4] "par"                  "par2"                "pair.logLik"
#> [7] "logLik"               "isDiscPermutated"    "listSupportPermutated"
#> [10] "uscorePlusPermutated" "uscoreMinusPermutated" "vine_array_perm"
#> [13] "is_disc"              "ntrunc"
print(fit_forwardD$family)
#>      [,1] [,2] [,3] [,4] [,5] [,6] [,7]
#> [1,]    0    7    1    1   20    5   14
#> [2,]    0    0    4   14    5    5    1
#> [3,]    0    0    0    1   20    1    5
#> [4,]    0    0    0    0    7    1    1
#> [5,]    0    0    0    0    0    5    5
#> [6,]    0    0    0    0    0    0   14
#> [7,]    0    0    0    0    0    0    0
print(fit_forwardD$par)
#>      [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
#> [1,]    0 0.879569 0.4653722 0.3005095 1.7316988 -0.1674511 3.2150070
#> [2,]    0 0.000000 1.3861900 1.4393994 0.6553449 0.5650022 0.5650387
#> [3,]    0 0.000000 0.0000000 0.4917370 2.7533698 -0.1177807 2.2031855
#> [4,]    0 0.000000 0.0000000 0.0000000 0.5889781 -0.2009586 0.2637945
#> [5,]    0 0.000000 0.0000000 0.0000000 0.0000000 -0.9148226 1.2980270
#> [6,]    0 0.000000 0.0000000 0.0000000 0.0000000 0.0000000 1.1020544
#> [7,]    0 0.000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
print(fit_forwardD$par2)
#>      [,1]      [,2] [,3] [,4]      [,5] [,6] [,7]
#> [1,]    0 1.205918    0    0 0.9819371    0    0
#> [2,]    0 0.000000    0    0 0.0000000    0    0
#> [3,]    0 0.000000    0    0 0.6477212    0    0
#> [4,]    0 0.000000    0    0 1.0330054    0    0
#> [5,]    0 0.000000    0    0 0.0000000    0    0
#> [6,]    0 0.000000    0    0 0.0000000    0    0
#> [7,]    0 0.000000    0    0 0.0000000    0    0

```

forwardR

```

fit_forwardR <- CopReg( data = udata,
  vine_array = vine_objects[["forwardR"]]$vine_array,
  family_set = c(1,2,4,5,7,10,14,17,20),
  is_disc = c(T, T, T, F, F, F, F) )
names(fit_forwardR)
#> [1] "vine_array_1tod"      "vine_array"          "family"
#> [4] "par"                  "par2"                "pair.logLik"
#> [7] "logLik"               "isDiscPermutated"    "listSupportPermutated"
#> [10] "uscorePlusPermutated" "uscoreMinusPermutated" "vine_array_perm"
#> [13] "is_disc"              "ntrunc"
print(fit_forwardR$family)
#>      [,1] [,2] [,3] [,4] [,5] [,6] [,7]
#> [1,]    0    7    1    1   20    1   14
#> [2,]    0    0    5   20    5    1    1
#> [3,]    0    0    0    1    1   20   17
#> [4,]    0    0    0    0    1    5    1

```

```

#> [5,] 0 0 0 0 0 14 5
#> [6,] 0 0 0 0 0 0 1
#> [7,] 0 0 0 0 0 0 0
print(fit_forwardR$par)
#>      [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
#> [1,] 0 0.879569 0.5861415 0.6640106 2.39475507 -0.15208540 3.2150070
#> [2,] 0 0.000000 1.0169183 1.4494470 1.05648040 -0.03536503 0.5650387
#> [3,] 0 0.000000 0.0000000 -0.1783439 -0.23315090 1.49398199 0.4943334
#> [4,] 0 0.000000 0.0000000 0.0000000 -0.07815636 0.07043382 0.2568806
#> [5,] 0 0.000000 0.0000000 0.0000000 0.00000000 1.06023186 1.4176793
#> [6,] 0 0.000000 0.0000000 0.0000000 0.00000000 0.00000000 0.1999800
#> [7,] 0 0.000000 0.0000000 0.0000000 0.00000000 0.00000000 0.0000000
print(fit_forwardR$par2)
#>      [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
#> [1,] 0 1.205918 0 0.0000000 0.97773195 0.0000000 0.0000000
#> [2,] 0 0.000000 0 0.8734744 0.0000000 0.0000000 0.0000000
#> [3,] 0 0.000000 0 0.0000000 0.0000000 0.9388814 1.035817
#> [4,] 0 0.000000 0 0.0000000 0.0000000 0.0000000 0.0000000
#> [5,] 0 0.000000 0 0.0000000 0.0000000 0.0000000 0.0000000
#> [6,] 0 0.000000 0 0.0000000 0.0000000 0.0000000 0.0000000
#> [7,] 0 0.000000 0 0.0000000 0.0000000 0.0000000 0.0000000

```

MSTx

```

fit_MSTx <- CopReg( data = udata,
  vine_array = vine_objects[["MSTx"]$vine_array,
  family_set = c(1,2,4,5,7,10,14,17,20),
  is_disc = c(T, T, T, F, F, F, F) )
names(fit_MSTx)
#> [1] "vine_array_1tod"      "vine_array"      "family"
#> [4] "par"                    "par2"            "pair.logLik"
#> [7] "logLik"                 "isDiscPermutated" "listSupportPermutated"
#> [10] "uscorePlusPermuted"     "uscoreMinusPermuted" "vine_array_perm"
#> [13] "is_disc"                "ntrunc"
print(fit_MSTx$family)
#>      [,1] [,2] [,3] [,4] [,5] [,6] [,7]
#> [1,] 0 1 20 7 1 1 14
#> [2,] 0 0 1 5 1 20 1
#> [3,] 0 0 0 5 14 20 1
#> [4,] 0 0 0 0 1 1 14
#> [5,] 0 0 0 0 0 1 5
#> [6,] 0 0 0 0 0 0 17
#> [7,] 0 0 0 0 0 0 0
print(fit_MSTx$par)
#>      [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
#> [1,] 0 0.5861415 2.3947551 0.879569 -0.15208540 0.66401060 3.2150070
#> [2,] 0 0.0000000 -0.2019067 1.056480 -0.03536503 1.44944703 0.5650387
#> [3,] 0 0.0000000 0.0000000 1.229824 1.04628980 1.49398199 0.1955595
#> [4,] 0 0.0000000 0.0000000 0.000000 -0.01685740 -0.06067482 1.1163638
#> [5,] 0 0.0000000 0.0000000 0.000000 0.00000000 -0.19112405 0.9058050
#> [6,] 0 0.0000000 0.0000000 0.000000 0.00000000 0.00000000 0.6389201
#> [7,] 0 0.0000000 0.0000000 0.000000 0.00000000 0.00000000 0.0000000
print(fit_MSTx$par2)

```

```

#>      [,1] [,2]      [,3]      [,4] [,5]      [,6]      [,7]
#> [1,]    0    0 0.9773195 1.205918    0 0.0000000 0.0000000
#> [2,]    0    0 0.0000000 0.000000    0 0.8734744 0.0000000
#> [3,]    0    0 0.0000000 0.000000    0 0.9388814 0.0000000
#> [4,]    0    0 0.0000000 0.000000    0 0.0000000 0.0000000
#> [5,]    0    0 0.0000000 0.000000    0 0.0000000 0.0000000
#> [6,]    0    0 0.0000000 0.000000    0 0.0000000 1.051058
#> [7,]    0    0 0.0000000 0.000000    0 0.0000000 0.000000

```

In-sample prediction

```

# quantile function of u_sweight
qdf <- function(p) { qsn(p, dp = coef(model_sweight, "DP")) }

insample_pred_forwardD <- predict( fit_forwardD,
  udata[,1:6], y_marginal_qfunc = qdf,
  interval = TRUE, level = 0.8)

insample_pred_forwardR <- predict( fit_forwardR,
  udata[,1:6], y_marginal_qfunc = qdf,
  interval = TRUE, level = 0.8)

insample_pred_MSTx <- predict( fit_MSTx,
  udata[,1:6], y_marginal_qfunc = qdf,
  interval = TRUE, level = 0.8)

```

Interval score function

```

#' @description
#' Interval score for prediction intervals
#'
#' @param predObj matrix with 3 columns: pointprediction, predLB, predUB
#' @param actual corresponding vector of actual values
#'           (in holdout set, for example)
#' @param level level for prediction interval, e.g., 0.5 or 0.8
#'
#' @return summary consisting of level, average length, interval score, coverage rate
#'
intervalScore <- function(predObj,actual,level)
{ n <- nrow(predObj)
  alpha <- 1-level
  ilow <- (actual<predObj[,2]) # overestimation
  ihigh <- (actual>predObj[,3]) # underestimation
  sumlength <- sum(predObj[,3]-predObj[,2]) # sum of lengths of prediction intervals
  sumlow <- sum(predObj[ilow,2]-actual[ilow])*2/alpha
  sumhigh <- sum(actual[ihigh]-predObj[ihigh,3])*2/alpha
  avglength <- sumlength/n
  IS <- (sumlength+sumlow+sumhigh)/n # average length + average under/over penalties
  cover <- mean(actual>= predObj[,2] & actual<=predObj[,3])
  summ <- c(level,avglength,IS,cover)
  names(summ) <- c("level","avglength","IS","coverage")
  return(summ)
}

```

```

in_score_forwardD <- intervalScore(insample_pred_forwardD, donkey.tr[, 7], 0.8)
print(in_score_forwardD)
#>      level avlength      IS coverage
#> 0.8000000 0.2083429 0.2877619 0.8000000
in_score_forwardR <- intervalScore(insample_pred_forwardR, donkey.tr[, 7], 0.8)
print(in_score_forwardR)
#>      level avlength      IS coverage
#> 0.8000000 0.2058604 0.2876622 0.7875000
in_score_MSTx <- intervalScore(insample_pred_MSTx, donkey.tr[, 7], 0.8)
print(in_score_MSTx)
#>      level avlength      IS coverage
#> 0.8000000 0.2070704 0.2859420 0.7900000

```

Out-of-sample prediction for holdout set

Transform the holdout data to uniform margins

```

nhold <- nrow(donkey.ho) # 144
uhold <- matrix(0, nhold, 7)
#colnames(donkey.ho)
uhold[,1] <- bcs_ecdf(donkey.ho[,1])
uhold[,2] <- age_ecdf(donkey.ho[,2])
uhold[,3] <- sex_ecdf(donkey.ho[,3])
uhold[,4] <- psn(donkey.ho[,4], dp = coef(model_slength, "DP"))
uhold[,5] <- psn(donkey.ho[,5], dp = coef(model_sgirth, "DP"))
uhold[,6] <- psn(donkey.ho[,6], dp = coef(model_sheight, "DP"))
uhold[,7] <- psn(donkey.ho[,7], dp = coef(model_sweight, "DP"))
# clip the values to [1e-5, 1-1e-5]
uhold <- pmax(pmin(uhold, 1-1e-5), 1e-5)
colnames(uhold) = c("ubcs", "uage", "usex", "ulen", "ugir", "uhei", "uwei")
summary(uhold)
#>      ubcs      uage      usex      ulen
#> Min.   :0.0600   Min.   :0.0850   Min.   :0.4625   Min.   :0.0002425
#> 1st Qu.:0.3125   1st Qu.:0.3575   1st Qu.:0.4625   1st Qu.:0.2173534
#> Median :0.8675   Median :0.8425   Median :0.6200   Median :0.5325856
#> Mean    :0.7036   Mean    :0.6529   Mean    :0.7094   Mean    :0.4908218
#> 3rd Qu.:0.8675   3rd Qu.:0.8425   3rd Qu.:1.0000   3rd Qu.:0.7152117
#> Max.    :1.0000   Max.    :1.0000   Max.    :1.0000   Max.    :0.9977743
#>      ugir      uhei      uwei
#> Min.   :0.0005699   Min.   :0.0000578   Min.   :0.0008625
#> 1st Qu.:0.2935298   1st Qu.:0.2551155   1st Qu.:0.2669622
#> Median :0.5136920   Median :0.4391257   Median :0.5298993
#> Mean    :0.4914762   Mean    :0.4751252   Mean    :0.4991101
#> 3rd Qu.:0.6954898   3rd Qu.:0.7293246   3rd Qu.:0.7499131
#> Max.    :0.9946745   Max.    :0.9844888   Max.    :0.9879810

```

Out-of-sample prediction

```

pred_forwardD <- predict( fit_forwardD,
  uhold[,1:6], y_marginal_qfunc = qdf,
  interval = TRUE, level = 0.8)

```

```

pred_forwardR <- predict( fit_forwardR,
  uhold[,1:6], y_marginal_qfunc = qdf,
  interval = TRUE, level = 0.8)

pred_MSTx <- predict( fit_MSTx,
  uhold[,1:6], y_marginal_qfunc = qdf,
  interval = TRUE, level = 0.8)

score_forwardD <- intervalScore(pred_forwardD, donkey.ho[, 7], 0.8)
print(score_forwardD)
#>      level avglength      IS coverage
#> 0.8000000 0.2058067 0.2951731 0.8111888
score_forwardR <- intervalScore(pred_forwardR, donkey.ho[, 7], 0.8)
print(score_forwardR)
#>      level avglength      IS coverage
#> 0.8000000 0.2037526 0.3073493 0.7972028
score_MSTx <- intervalScore(pred_MSTx, donkey.ho[, 7], 0.8)
print(score_MSTx)
#>      level avglength      IS coverage
#> 0.8000000 0.2049934 0.3029165 0.7972028

```

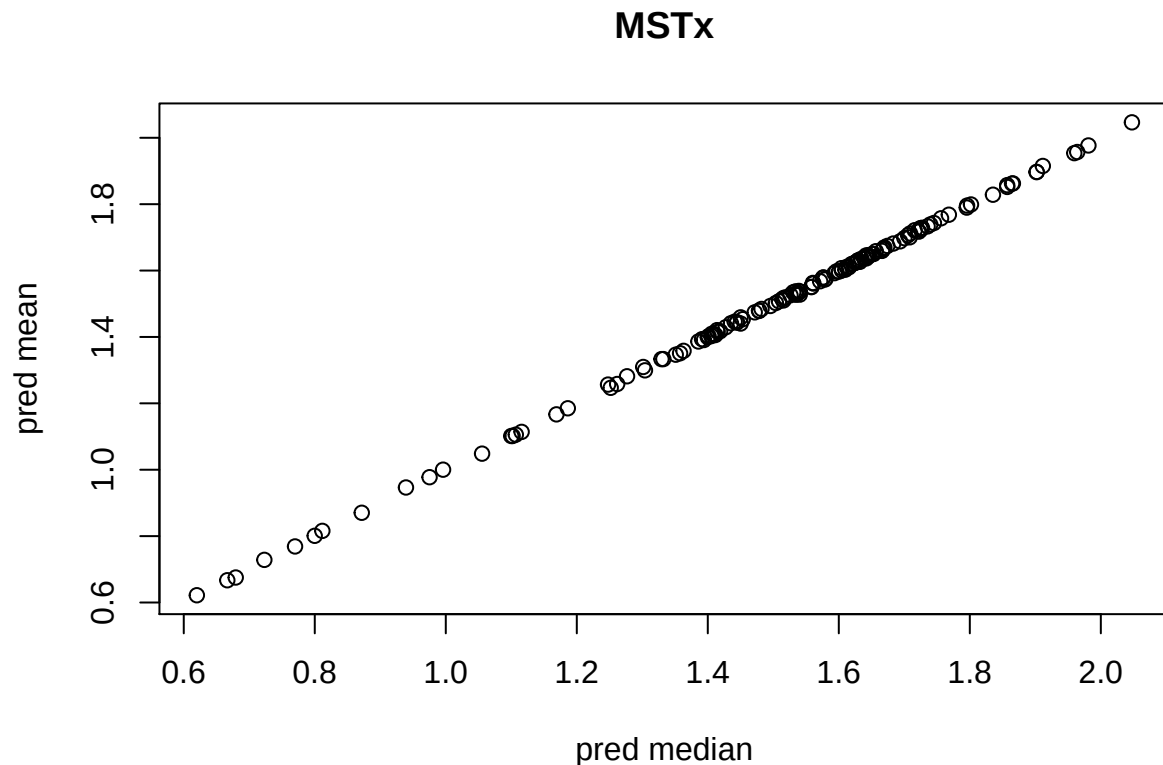
By default, the point estimate is the conditional median. Alternatively, one can use the conditional mean as the point estimate by setting `mean = TRUE`.

```

pred_MSTx_mean <- predict( fit_MSTx,
  uhold[,1:6], y_marginal_qfunc = qdf,
  mean = TRUE, interval = TRUE, level = 0.8)

# Plot of predicted median versus predicted mean for MSTx fit
plot(pred_MSTx[,1], pred_MSTx_mean[, 1], xlab = "pred median", ylab = "pred mean", main = "MSTx")

```



We also provide an option to compute the *predicted log-likelihood*, which can be used as a scoring rule, i.e., the logarithmic score. When `predict_llk` is set to `TRUE`, two additional parameters are required: `new_response` and `y_marginal_cdf`.

```
# cdf of u_sweight
y_cdf <- function(x) { psn(x, dp = coef(model_sweight, "DP")) }

pred_forwardD_llk <- predict( fit_forwardD,
  uhold[,1:6], y_marginal_qfunc = qdf,
  predict_llk = TRUE, new_response = donkey.ho[, 7],
  y_marginal_cdf = y_cdf )

pred_forwardR_llk <- predict( fit_forwardR,
  uhold[,1:6], y_marginal_qfunc = qdf,
  predict_llk = TRUE, new_response = donkey.ho[, 7],
  y_marginal_cdf = y_cdf )

pred_MSTx_llk <- predict( fit_MSTx,
  uhold[,1:6], y_marginal_qfunc = qdf,
  predict_llk = TRUE, new_response = donkey.ho[, 7],
  y_marginal_cdf = y_cdf )

# predicted loglikelihood
# larger is better, so forwardD is marginally better based on this out-of-sample log score.
print(pred_forwardD_llk$predicted_loglik)
#> [1] 1.09371
print(pred_forwardR_llk$predicted_loglik)
```

```

#> [1] 1.057508
print(pred_MSTx_llk$predicted_loglik)
#> [1] 1.059348

# predicted loglikelihood per sample for holdout set, larger is better.
head(pred_forwardD_llk$predicted_loglik_per_sample)
#> [1] 1.2297077 1.3164873 1.8197826 1.7606521 -0.4340541 0.6903102
head(pred_forwardR_llk$predicted_loglik_per_sample)
#> [1] 1.49335286 1.30547660 1.77225309 1.67746996 -0.00734841 1.95485151
head(pred_MSTx_llk$predicted_loglik_per_sample)
#> [1] 1.4602515 1.2561863 1.8125202 1.7261714 -0.2565096 1.7600971

```

Save the environment

```

save.image(file="donkey_vignette.RData")

```