LRMoE RealData Result

Spark Tseung

March 3, 2020

Introduction

This document is Part III of a demo series of the `LRMoE` (Logit-weighted Reduced Mixture-of-Experts) package on a real dataset. By analysing a French motor third-party liability insurance dataset in `CASdatasets`, we will demonstrate the fitting procedure, diagnostics, visualization and predictive functions of the `LRMoE` package. In this document, we demonstrate various package utilities for model selection, actuarial pricing and model visualization.

Fitting Result

Using the fitting code in Part II, we have obtained a collection of LRMoE models. We will use the best one `llblll` as an illustrative example.

```
# Load data from Part I
load("X.Rda")
load("Y.Rda")

# Load fitted model from Part II
load("1_llblll.Rda")
```

The model `llblll` is the *best* in the sense of maximising the Akaike Information Criterion (AIC) among all models we tried. The loglikelihood (with or without penalty), AIC, and BIC of the fitted model can be inspected using standard R methods.

```
# loglikelihood
model.fit$ll
```

```
## [1] -183524
```

```
model.fit$ll.np
```

```
## [1] -183444
```

```
# AIC
model.fit$AIC
```

```
## [1] 367226
```

```
# BIC
model.fit$BIC
```

```
## [1] 369073.5
```

Actuarial Pricing and Risk Measures

The `LRMoE` package contains a collection of functions related to actuarial pricing, reserving and risk management, including calculation of mean, variance, value at risk (VaR), conditional tail expectation (CTE), limited expected value (LEV) and stop-loss (SL) premium of the response variable. These functions start with root predict., followed by appropriate quantities of interest (mean, var, quantile, cte, limit, excess) and corresponding function arguments.

For example, consider policyholders 1, 33 and 96.

```
# Mean of claim amount of Policyholders A, B and C.
# Variance is infinite due to Burr component.
predict.mean(X[c(1, 33, 96),],
  model.fit$alpha.fit, model.fit$comp.dist,
  model.fit$zero.fit, model.fit$params.fit)
```

```
##          [,1]
## [1,] 97.48500
## [2,] 90.25394
## [3,] 84.64872
```

```
predict.var(X[c(1, 33, 96),],
  model.fit$alpha.fit, model.fit$comp.dist,
  model.fit$zero.fit, model.fit$params.fit)
```

```
##      [,1]
## [1,]  Inf
## [2,]  Inf
## [3,]  Inf
```

```
# 99% VaR of claim amount of Policyholders A, B and C.
predict.quantile(prob = 0.99, X[c(1, 33, 96),],
  model.fit$alpha.fit, model.fit$comp.dist,
  model.fit$zero.fit, model.fit$params.fit)
```

```
##          [,1]
## [1,] 1209.099
## [2,] 1216.505
## [3,] 1249.037
```

```
# SL premium (d=1000) of Policyholders A, B and C.
predict.excess(limit = 1000, X[c(1, 33, 96),],
  model.fit$alpha.fit, model.fit$comp.dist,
  model.fit$zero.fit, model.fit$params.fit)
```

```
##          [,1]
## [1,] 78.75538
## [2,] 69.81123
## [3,] 58.28566
```

```
# LEV of claim amount (d=100000) of Policyholders A, B and C.
predict.limit(limit = 100000, X[c(1, 33, 96),],
  model.fit$alpha.fit, model.fit$comp.dist,
  model.fit$zero.fit, model.fit$params.fit)
```

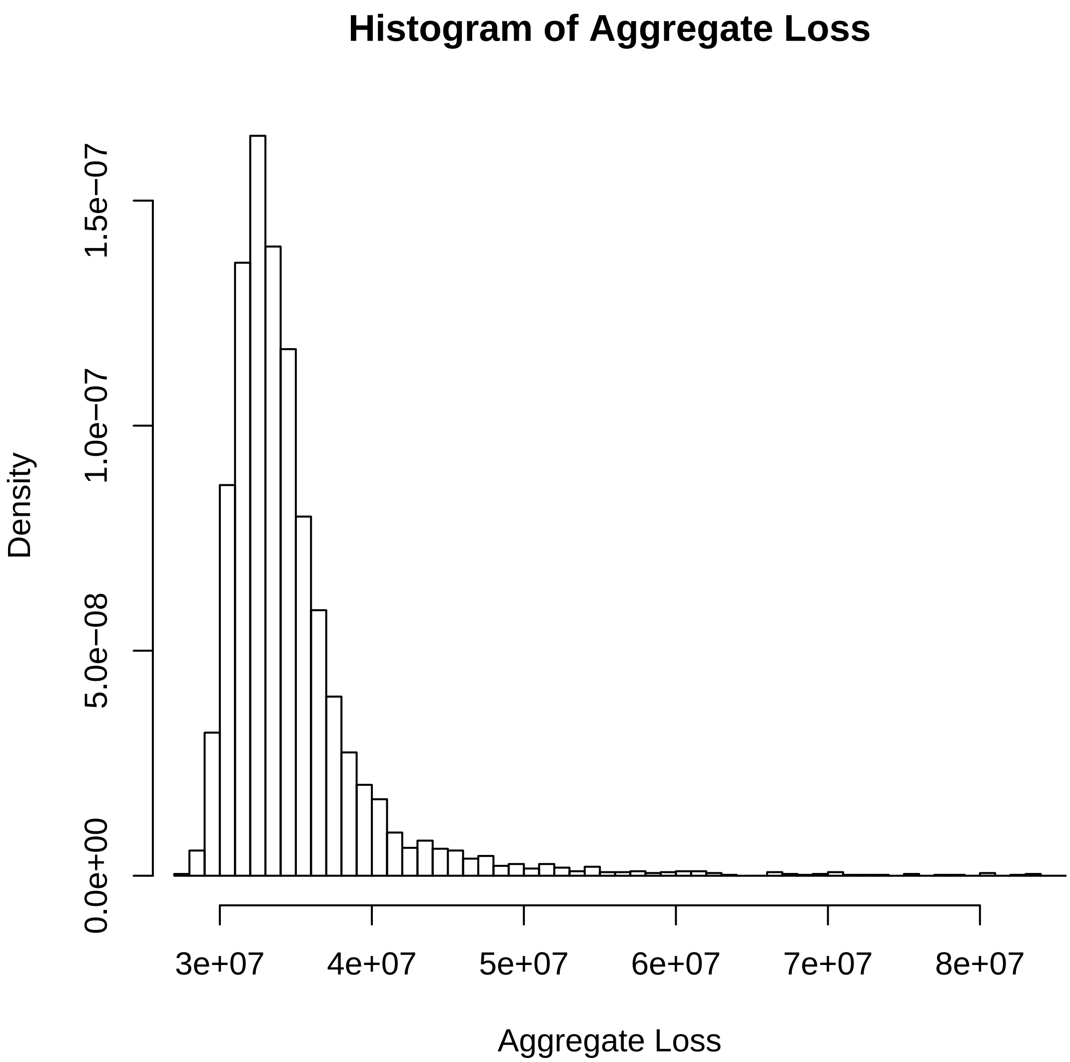
```
##          [,1]
## [1,] 63.23804
## [2,] 60.85653
## [3,] 63.32523
```

At a portfolio level, we can simulate the distribution of the aggregate loss, which can be useful for setting the insurer's reserve, as well as allocated back to policyholders as a loaded premium. The simulation has been done with the `dataset.simulator` function in a separate file. (*Note: Run in parallel of 10 processes, the simulation of 5000 scenarios takes about 3 hours.*)

```
# Load simulated aggregated loss
nsim = 5000
ngroup = 100

result = rep(NA, nsim)

# Each simtable_j contains 50 scenarios
for(j in 1:ngroup)
{
  filename = toString(paste("./3-SimAggreLoss/simtable_", j, ".Rda", sep = ""))
  load(filename)
  # temp.agg = apply(sim.table, 1, FUN = sum)
  result[c((50*(j-1)):(50*(j-1)+49))+1] = apply(sim.table, 1, FUN = sum)
  rm(sim.table)
}

# Summary of simulated aggregate loss
summary(result)
```

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## 2.782e+07 3.194e+07 3.353e+07 3.556e+07 3.583e+07 1.562e+09
```

The histogram of the aggregate loss is shown below, which is quite different from the loss distribution of each individual policyholder.



For illustration, we can use the expected claim amount to weight each policyholder, and allocate some metric (e.g. VaR, CTE) of the aggregate loss as a loaded premium.

```
# Policyholder's weights
pred.mean = predict.mean(X,
    model.fit$alpha.fit, model.fit$comp.dist,
    model.fit$zero.fit, model.fit$params.fit)

weighting = sweep(as.matrix(pred.mean), 2,
    STATS = sum(pred.mean), FUN = "/", check.margin = FALSE)
```

```
# Calculate various quantities of interest
# Mean
meanResult = mean(result)

# SD
sdResult = sqrt(var(result))

# VAR
VAR700 = quantile(result, 0.70)
VAR800 = quantile(result, 0.80)
VAR900 = quantile(result, 0.90)
VAR950 = quantile(result, 0.95)
VAR990 = quantile(result, 0.99)

# CTE
CTE700 = mean(result[which(result>VAR700)])
CTE800 = mean(result[which(result>VAR800)])
CTE900 = mean(result[which(result>VAR900)])
CTE950 = mean(result[which(result>VAR950)])
CTE990 = mean(result[which(result>VAR990)])
```

```
# Allocate back to policyholders as premium

price.mean = sweep(as.matrix(weighting), 1,
                   STATS = meanResult, FUN = "*", check.margin = TRUE)

price.SD.50 = sweep(as.matrix(weighting), 1,
                    STATS = meanResult+0.5*sdResult, FUN = "*", check.margin = TRUE)
price.SD.75 = sweep(as.matrix(weighting), 1,
                    STATS = meanResult+0.75*sdResult, FUN = "*", check.margin = TRUE)
price.SD.00 = sweep(as.matrix(weighting), 1,
                    STATS = meanResult+1*sdResult, FUN = "*", check.margin = TRUE)

price.VAR700 = sweep(as.matrix(weighting), 1,
                     STATS = VAR700, FUN = "*", check.margin = TRUE)
price.VAR900 = sweep(as.matrix(weighting), 1,
                     STATS = VAR900, FUN = "*", check.margin = TRUE)
price.VAR950 = sweep(as.matrix(weighting), 1,
                     STATS = VAR950, FUN = "*", check.margin = TRUE)
price.VAR990 = sweep(as.matrix(weighting), 1,
                     STATS = VAR990, FUN = "*", check.margin = TRUE)

price.CTE700 = sweep(as.matrix(weighting), 1,
                     STATS = CTE700, FUN = "*", check.margin = TRUE)
```

```r
price.CTE800 = sweep(as.matrix(weighting), 1,
                     STATS = CTE800, FUN = "*", check.margin = TRUE)
price.CTE900 = sweep(as.matrix(weighting), 1,
                     STATS = CTE900, FUN = "*", check.margin = TRUE)
price.CTE950 = sweep(as.matrix(weighting), 1,
                     STATS = CTE950, FUN = "*", check.margin = TRUE)
price.CTE990 = sweep(as.matrix(weighting), 1,
                     STATS = CTE990, FUN = "*", check.margin = TRUE)

df = data.frame(pred.mean,
                price.mean, price.SD.50, price.SD.75, price.SD.00,
                price.VAR700, price.VAR900, price.VAR950, price.VAR990,
                price.CTE700, price.CTE800, price.CTE900, price.CTE950, price.CTE990)
```

Again, consider policyholders 1, 33 and 96. Notice the first two rows `pred.mean` and `price.mean` are theoretically equal, but differ a little bit due to simulation noise.

```r
t(df[c(1, 33, 96),])
```

```
##                      1        33        96
## pred.mean     97.48500  90.25394  84.64872
## price.mean    97.63874  90.39628  84.78223
## price.SD.50  130.33054 120.66313 113.16935
## price.SD.75  146.67644 135.79656 127.36292
## price.SD.00  163.02234 150.92998 141.55648
## price.VAR700  96.62205  89.45500  83.89940
## price.VAR900 108.64638 100.58742  94.34044
## price.VAR950 121.13614 112.15074 105.18562
## price.VAR990 183.38590 169.78305 159.23868
## price.CTE700 117.20247 108.50885 101.76992
## price.CTE800 126.57824 117.18916 109.91114
## price.CTE900 149.27708 138.20429 129.62112
## price.CTE950 184.98653 171.26495 160.62855
## price.CTE990 370.97346 343.45611 322.12578
```

Model Visualization

The `LRMoE` package contains some built-in visualization tools for predicting the latent class probabilities (or proportion) for each policyholder (or for the entire dataset). The `data.simulator` function also helps creating more customized plots.

Latent Class Probabilities

The probability of latent classes can be calculated using the `predict.` function, and visualized by `plot.ind.posterior.prob`.

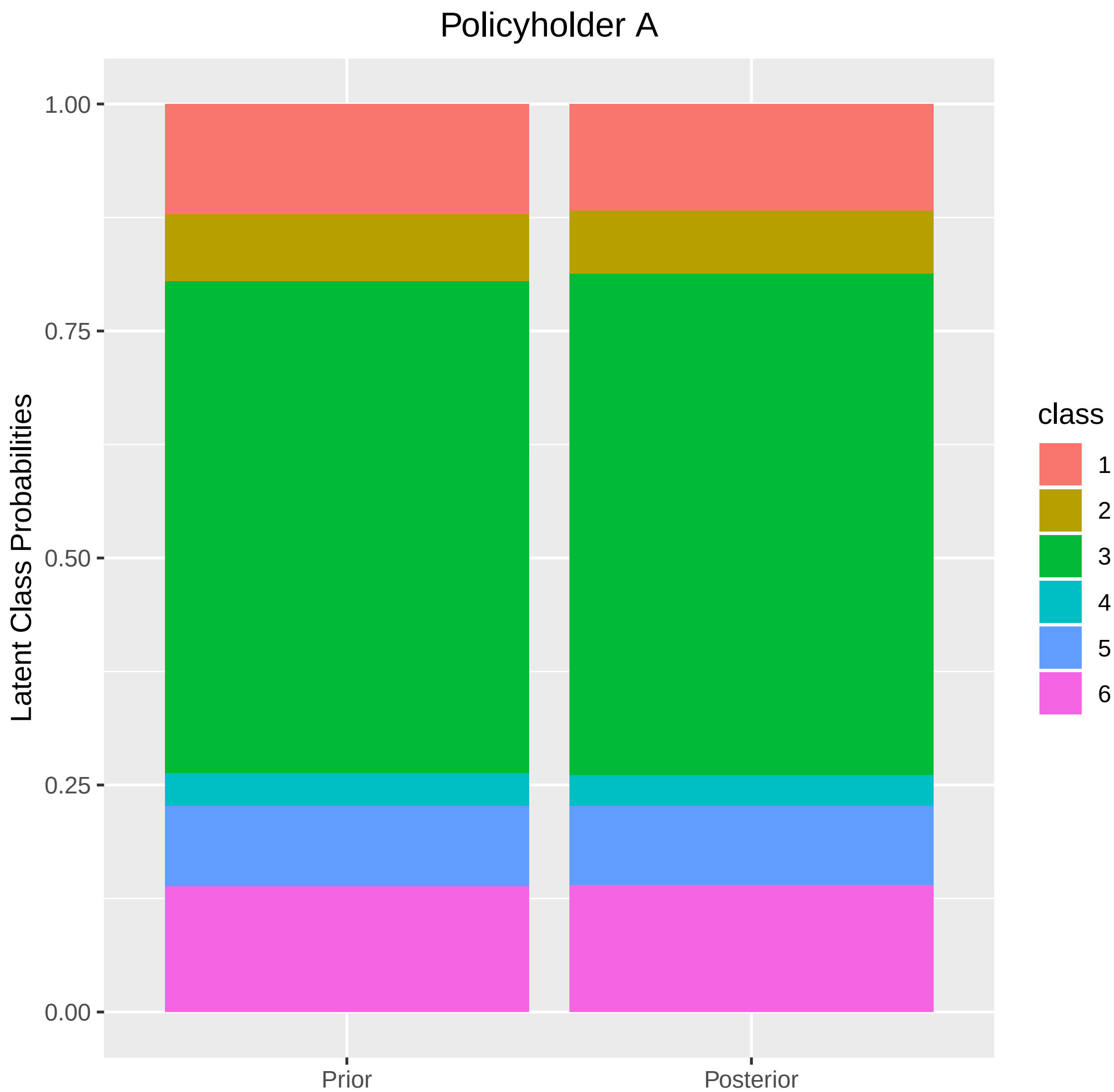
```
# Predict latent class probabilities, based on covariates and a model
predict.class.prob(X[c(1,33,96),], model.fit$alpha.fit)
```

```
##         comp 1     comp 2    comp 3     comp 4     comp 5    comp 6
## [1,] 0.1213162 0.07381480 0.5416097 0.03613851 0.08855876 0.1385620
## [2,] 0.1404270 0.07858723 0.4647998 0.04097918 0.08278394 0.1924228
## [3,] 0.2082516 0.10479211 0.3364129 0.04198693 0.10295946 0.2055970
```

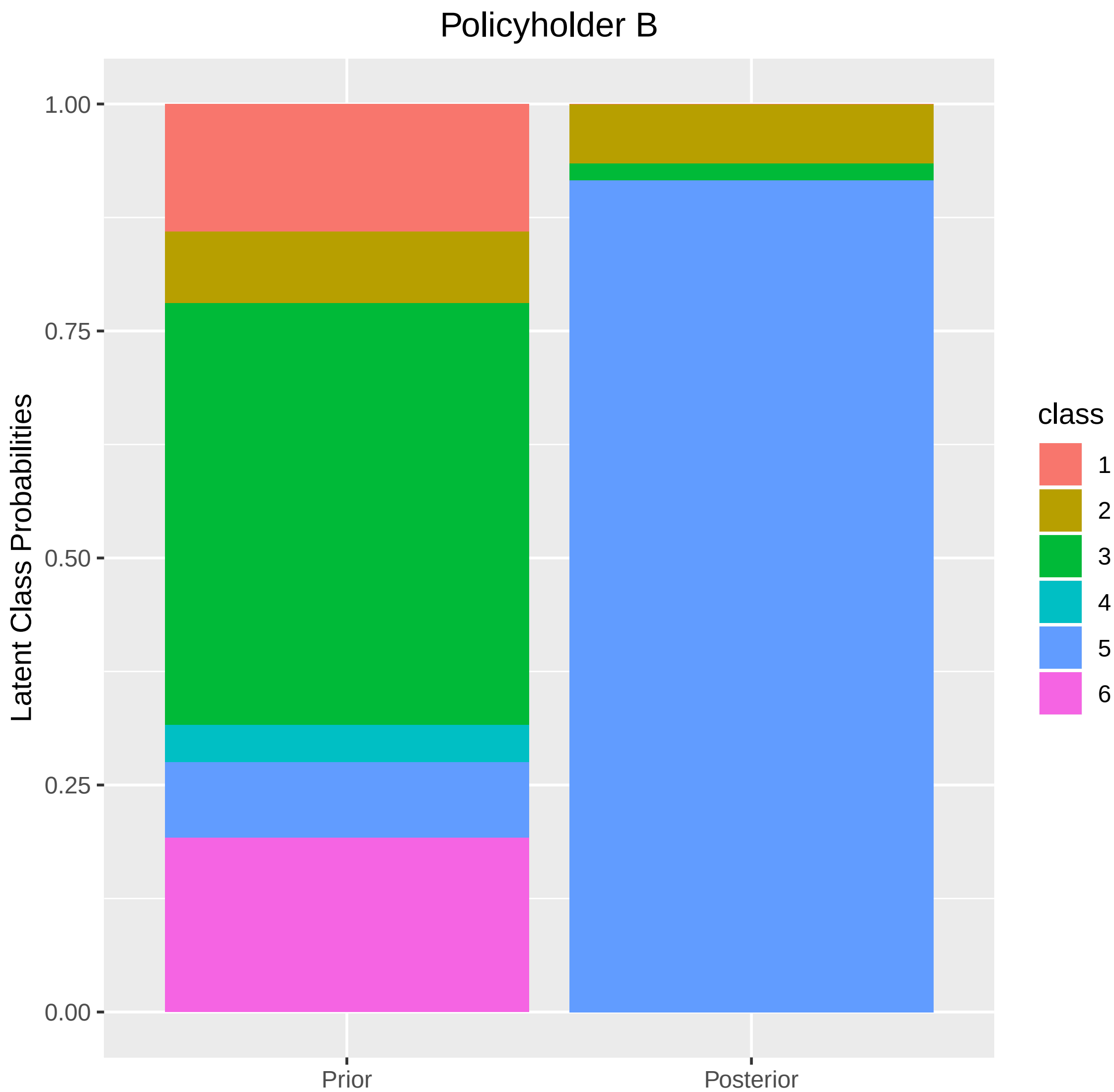
```
# Predict posterior probabilities, based on covariates, history and a model
predict.class.prob.posterior(X[c(1,33,96),], Y[c(1,33,96),],
  model.fit$alpha.fit, model.fit$comp.dist,
  model.fit$zero.fit, model.fit$params.fit)
```

```
##             comp 1     comp 2    comp 3        comp 4     comp 5       comp 6
## [1,]  1.174421e-01 0.06931930 0.5521331  3.399789e-02 0.08667928 1.404283e-01
## [2,] 2.577365e-294 0.06546023 0.0185750 1.509372e-230 0.91596478 8.194626e-25
## [3,]  0.000000e+00 0.12230349 0.4648738  0.000000e+00 0.41282273 0.000000e+00
```

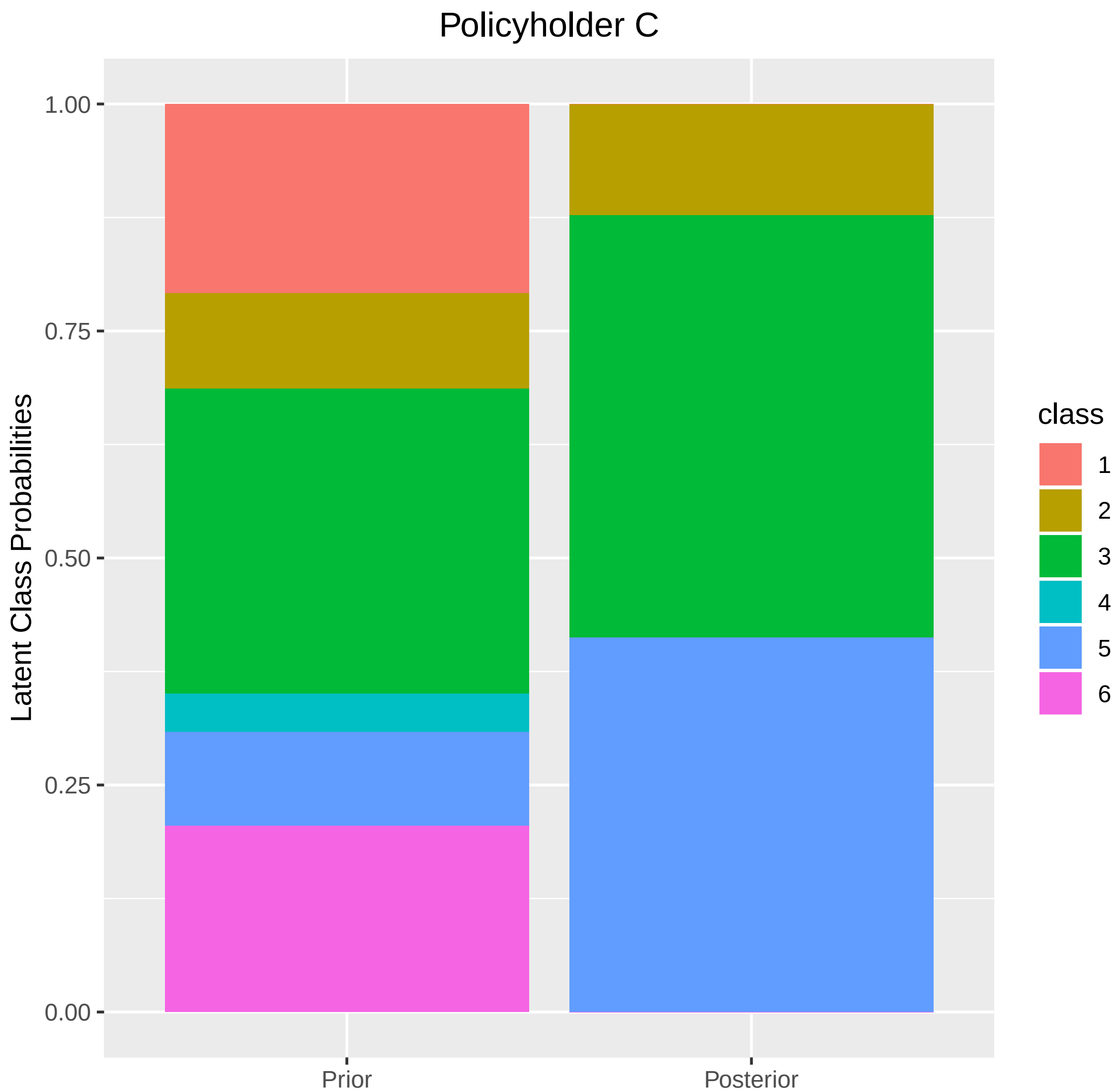
```r
# Plot latent class probabilities for Policyholder A
plot.ind.class.prob.posterior(X[1,], Y[1,], model.fit$alpha.fit, model.fit$comp.dist,
  model.fit$zero.fit, model.fit$params.fit, title = "Policyholder A")
```



```r
# Plot latent class probabilities for Policyholder B
plot.ind.class.prob.posterior(X[33,], Y[33,], model.fit$alpha.fit, model.fit$comp.dist,
  model.fit$zero.fit, model.fit$params.fit, title = "Policyholder B")
```

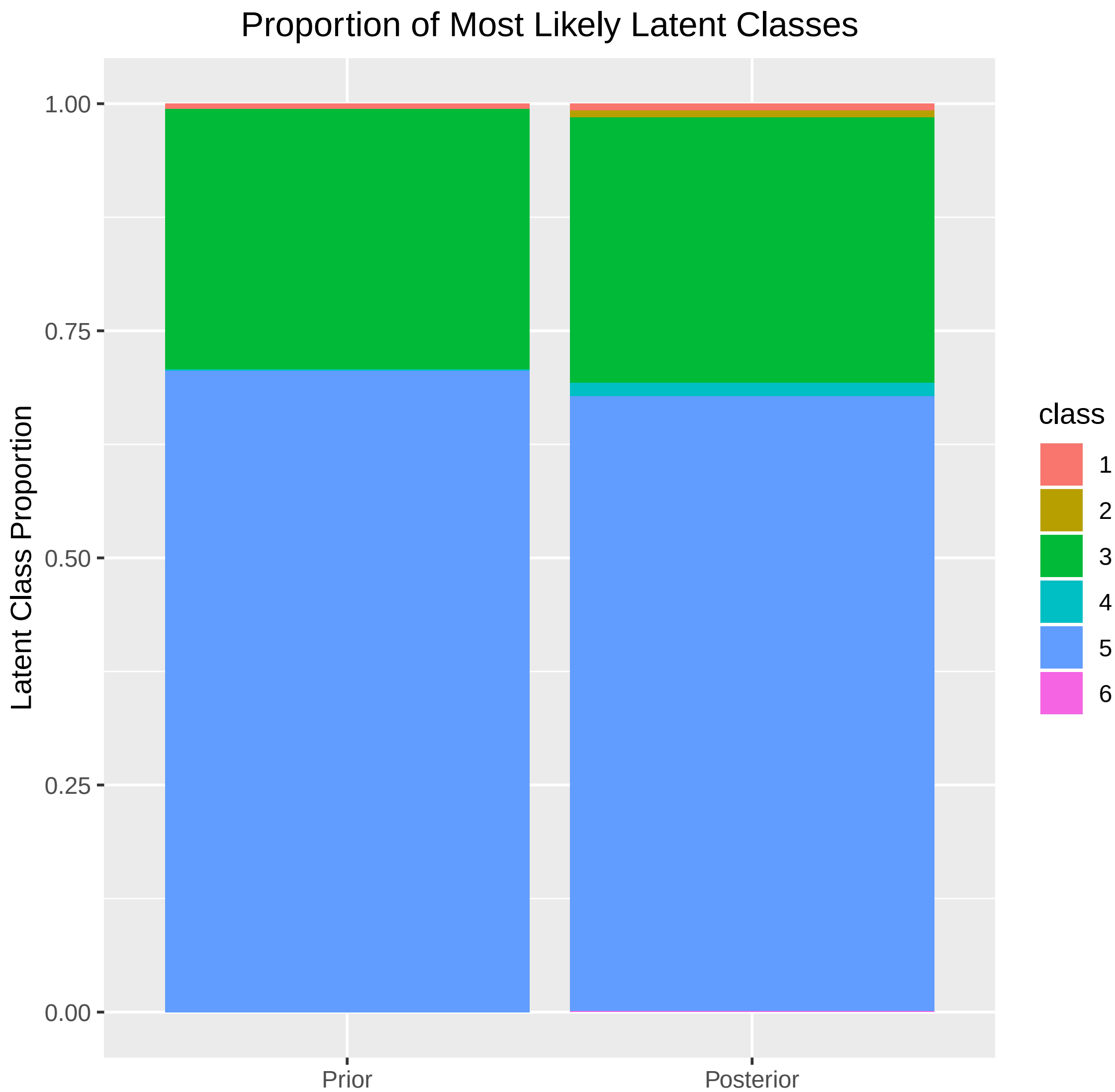


```r
# Plot latent class probabilities for Policyholder C
plot.ind.class.prob.posterior(X[96,], Y[96,], model.fit$alpha.fit, model.fit$comp.dist,
  model.fit$zero.fit, model.fit$params.fit, title = "Policyholder C")
```



The same can be plotted for the entire dataset. Instead of the probabilities of latent classes, the most likely class is predicted for each policyholder, and the proportion of most likely classes are plotted.

```
# Plot most likely classes for the entire dataset
plot.dataset.prob.posterior(X, Y, model.fit$alpha.fit, model.fit$comp.dist,
  model.fit$zero.fit, model.fit$params.fit,
  title = "Proportion of Most Likely Latent Classes")
```



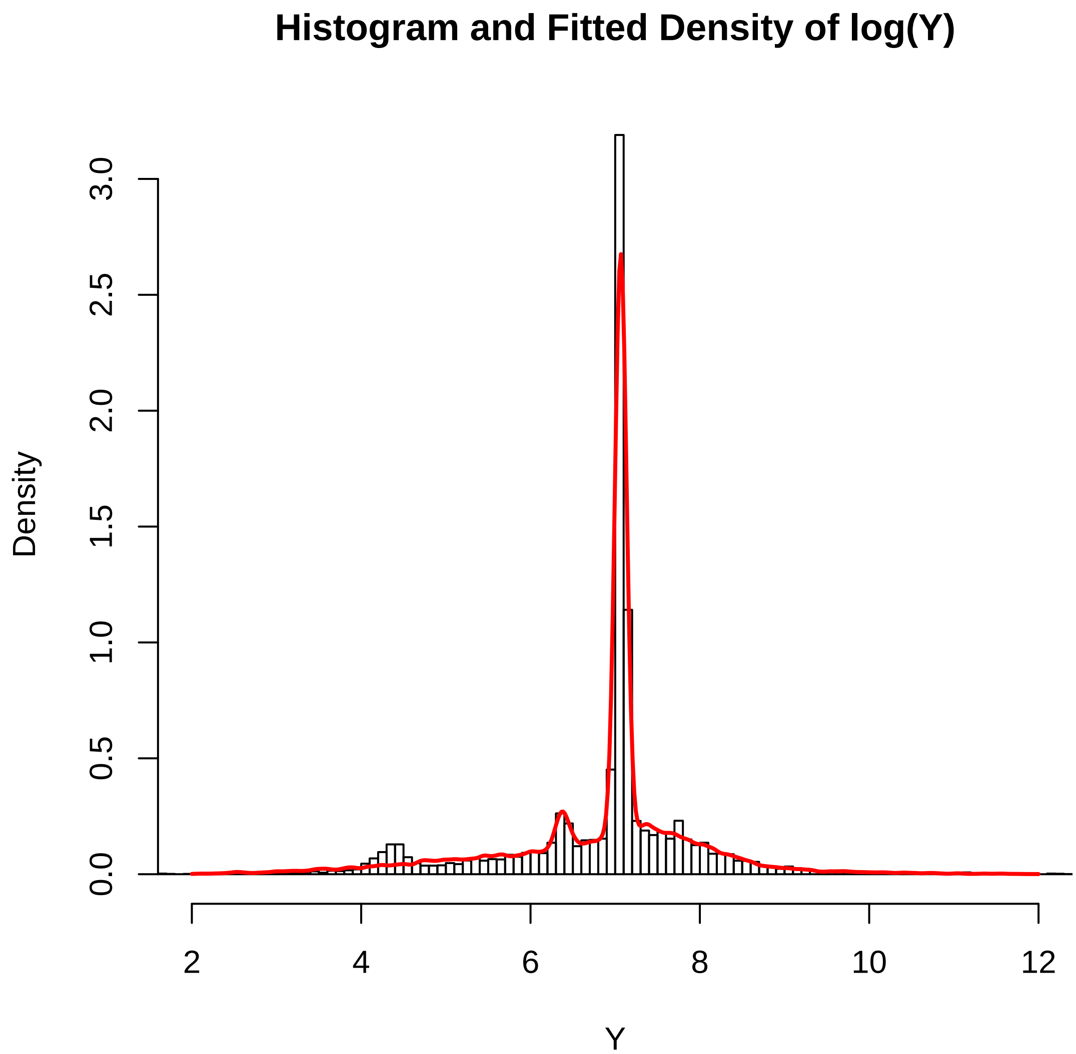
Overall Goodness-of-Fit

The overall goodness-of-fit can be examined by plotting either the fitted density against the histogram of data, or the QQ plot.

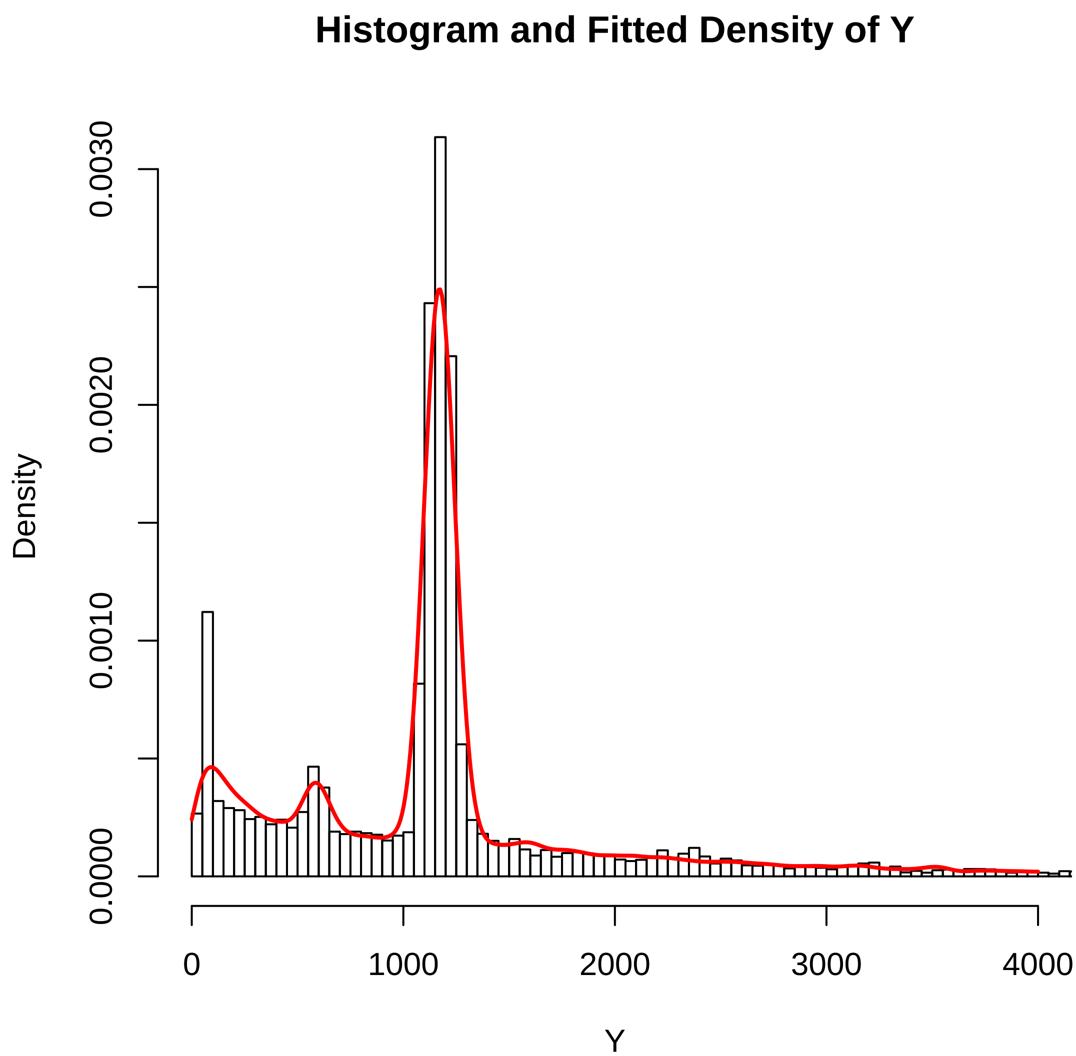
For reasons explained in Part I, we use simulation for noth the fitted density and the QQ plot.

```r
# Simulate exact responses, given a set of covariates and a fitted model
set.seed(77)
sim.size = nrow(X)
model.sim = dataset.simulator(X, model.fit$alpha.fit, model.fit$comp.dist,
    model.fit$zero.fit, model.fit$params.fit)

# Only use positive values for plotting density
Y.pos = Y[which(Y[,2]>0),2]
sim.Y.pos = model.sim[which(model.sim[,1]>0),1]
```

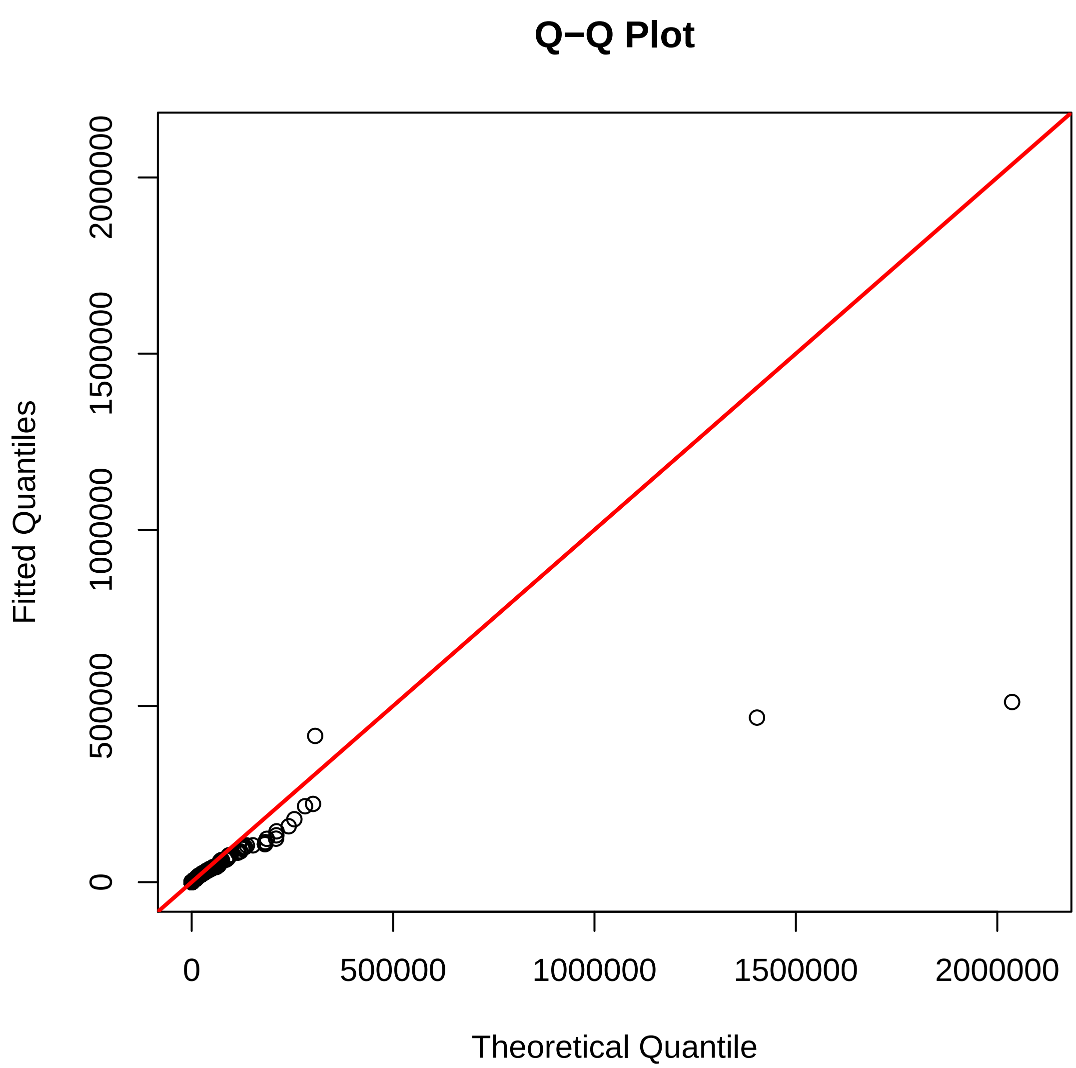
```r
# Use standard R functions for plotting histograms (all data points)
hist(log(Y.pos), breaks = 100, xlim = c(2, 12), probability = TRUE,
     xlab = "Y", main = "Histogram and Fitted Density of log(Y)")
lines(density(log(sim.Y.pos), from = 2, to = 12), col = "red", lwd = 2)
```


Histogram and Fitted Density of log(Y)
Density
0.0
0.5
1.0
1.5
2.0
2.5
3.0
2
4
6
8
10
12
Y

```r
# Use standard R functions for plotting histograms (non-extreme values)
hist(Y.pos, breaks = 50000, xlim = c(0, 4000), probability = TRUE,
     xlab = "Y", main = "Histogram and Fitted Density of Y")
lines(density(sim.Y.pos, from = 0, to = 4000), col = "red", lwd = 2)
```



```
# Use standard R functions for Q-Q plots (all data points)

# qqplot(Y[,2], model.sim[,1], main = "Q-Q Plot",
#   xlab = "Theoretical Quantile", ylab = "Fitted Quantiles")
# abline(a = 0, b = 1, col = "red", lwd = 2)

plot(range(0, 2100000), range(0, 2100000), type = "n",
     xlab = "Theoretical Quantile", ylab = "Fitted Quantiles", main = "Q-Q Plot")
QQ.model = qqplot(Y[,2], model.sim, plot.it = FALSE)
points(QQ.model, pch = 1)
abline(a=0,b=1,col="red", lwd = 2)
```



```r
# Use standard R functions for Q-Q plots (non-extreme values)
qqplot(Y[,2], model.sim,
       xlim = c(0, 20000), ylim = c(0, 20000),
       xlab = "Theoretical Quantile", ylab = "Fitted Quantiles", main = "Q-Q Plot")
abline(a=0,b=1,col="red", lwd = 2)
```