LRMoE DemoData Fitting

Spark Tseung

March 2, 2020

Introduction

This document contains the data fitting process for the dataset `DemoData` included in the `LRMoE` package. This serves as an example of using the main fitting function `LRMoE.fit` included in the package.

Data Loading

The `DemoData` in the package can be loaded as follows. The data generation process has been described in a previous document.

```
data("DemoData")
```

Fitting `LRMoE`

In this section, we demonstrate how to fit an LRMoE model in the package. In the current version of `LRMoE`, the minimal inputs required from the user are: response, covariates, number of component distributions to use, specification of component distributions, initial guesses of parameters.

Correctly Specified Model

We first start with a correctly specified LRMoE with parameter guesses close to the true ones, which aims to show that the package can identify the true model when the component distributions are correctly given along with reasonable guesses of parameters.

```
# Number of component distributions; Number of response dimension
n.comp = 2 # = g
dim.m =  2 # = d

# Specify component distributions
# by dimension (row) and by component (column)
# Dimension is d * g
comp.dist = matrix(c("poisson", "ZI-gammacount",
                     "lnorm", "invgauss"),
                   nrow = dim.m, byrow = TRUE)

# Initial guesses of alpha: logit regression weights
# Dimension is g * P
alpha.init = matrix( c(0, 0, 0, 0, 0,
                        0, 0, 0, 0, 0),
                      nrow = n.comp, byrow = TRUE)

# Initial guesses of zero-inflation probabilities
# Dimension is d * g
zero.init = matrix(c(0, 0.5, # ),
```

```r
                     0, 0),
                  nrow = dim.m, byrow = TRUE)

# Initial guesses of component distribution parameters
# d-length list, where each elememtn is a g-length list of vectors
params.init = list( list(c(10), c(40, 0.8)),
                    list(c(3, 1), c(15, 15))
                    )

# Penalty for alpha: a single numeric for all logit regression weights
hyper.alpha = 5

# Penalty for component distribution parameters
# Dimension is the same as params.guess
hyper.params = list( list(c(1, Inf), c(9, 0.5, 9, 0.5)),
                     list(c(Inf, 1, Inf), c(9, 0.5, 9, 0.5))
                    )
```

Now we are ready to call the fitting function. It is optional to print out intermediate updates of parameters. (*Note: The fitting function takes about 5 minutes to run.*)

```r
fitted.model = LRMoEprivate::LRMoE.fit(Y = Y.obs, X = X.obs, n.comp = n.comp,
                 comp.dist = comp.dist,
                 alpha.init = alpha.init,
                 zero.init = zero.init, params.init = params.init,
                 penalty = TRUE,
                 hyper.alpha = hyper.alpha, hyper.params = hyper.params,
                 print = FALSE
                 )
```

The fitting function will return a list of updated parameters, as well as the loglikelihood, Akaike Information Criterion (AIC) and Bayesian Information Criterion (BIC) of the fitted model. We observe that the fitted model is reasonbly close to the true model, considering simulation errors and loss of information due to data truncation and censoring.

```r
# Fitted logit regression weights
fitted.model$alpha.fit
```

```
##          intercept       sex    agedriver     agecar   region
## comp 1 -0.3926946 0.9836609 -0.04925043 0.09100969 1.152698
## comp 2  0.0000000 0.0000000  0.00000000 0.00000000 0.000000
```

```r
# Fitted zero-inflation probabilities
fitted.model$zero.fit
```

```
##      [,1]      [,2]
## [1,]    0 0.1976008
## [2,]    0 0.0000000
```

```r
# Fitted parameters of component distributions
fitted.model$params.fit
```

```
## $`dim 1`
## $`dim 1`$`comp 1`
##     mean
## 5.866834
##
```

```
## $`dim 1`$`comp 2`
##          m          s
## 29.5159775  0.4923445
##
##
## $`dim 2`
## $`dim 2`$`comp 1`
##   meanlog     sdlog
## 4.0096657 0.3009845
##
## $`dim 2`$`comp 2`
##     mean    scale
## 19.92436 22.04913
```

```
# Loglikelihood: with and without parameter penalty
fitted.model$ll
```

```
## [1] -72924.14
```

```
fitted.model$ll.np
```

```
## [1] -72885.48
```

```
# AIC
fitted.model$AIC
```

```
## [1] 145797
```

```
# BIC
fitted.model$BIC
```

```
## [1] 145890.5
```

Mis-Specified Model

In practice, it is almost impossible to know the **true** underlying distribution of data. Assume the user has conducted some preliminary analysis, and proposes to use the following LRMoE.

```
# Number of component distributions; Number of response dimension
n.comp = 2 # = g
dim.m = 2 # = d

# Specify component distributions
comp.dist = matrix(c("ZI-poisson", "ZI-nbinom",
                     "burr", "gamma"),
                   nrow = dim.m, byrow = TRUE)

# Initial guesses of alpha: logit regression weights
alpha.guess = matrix( c(0, 0, 0, 0, 0,
                        0, 0, 0, 0, 0),
                      nrow = n.comp, byrow = TRUE)

# Initial guesses of zero-inflation probabilities
zero.guess = matrix(c(0.5, 0.5,
                      0, 0),
                   nrow = dim.m, byrow = TRUE)

# Initial guesses of component distribution parameters
```

```
params.guess = list( list(c(10), c(25, 0.4)),
                     list(c(5, 2, 30), c(1, 10))
                    )

# Penalty for alpha: a single numeric for all logit regression weights
hyper.alpha = 5

# Penalty for component distribution parameters
# Dimension is the same as params.guess
hyper.params = list( list(c(5, 5), c(5, 5)),
                     list(c(5, 5, 5, 5, 5, 5), c(5, 5, 5, 5))
                    )
```

The fitting function can be called similarly. (*Note: The fitting function takes about 10 minutes to run, mostly due to numerical integration and optimization of the Burr component.*)

```
fitted.model.new = LRMoEprivate::LRMoE.fit(Y = Y.obs, X = X.obs, n.comp = n.comp,
                     comp.dist = comp.dist,
                     alpha.init = alpha.guess,
                     zero.init = zero.guess, params.init = params.guess,
                     penalty = TRUE,
                     hyper.alpha = hyper.alpha, hyper.params = hyper.params,
                     print = FALSE
                     )
```

We can also examine the mis-specified model. Judging from the loglikelihood, it provides relatively worse fit to data compared with the true model.

```
# Fitted logit regression weights
fitted.model.new$alpha.fit
```

```
##         intercept       sex   agedriver     agecar   region
## comp 1 -0.3674283 0.9810763 -0.04940752 0.09170515 1.148301
## comp 2  0.0000000 0.0000000  0.00000000 0.00000000 0.000000
```

```
# Fitted zero-inflation probabilities
fitted.model.new$zero.fit
```

```
##             [,1]      [,2]
## [1,] 0.008751151 0.1951587
## [2,] 0.000000000 0.0000000
```

```
# Fitted parameters of component distributions
fitted.model.new$params.fit
```

```
## $`dim 1`
## $`dim 1`$`comp 1`
##     mean
## 5.878512
##
## $`dim 1`$`comp 2`
##       size       prob
## 30.0000000  0.4994876
##
##
## $`dim 2`
## $`dim 2`$`comp 1`
##    shape1    shape2     scale
##  1.117386  5.565085 56.986149
##
## $`dim 2`$`comp 2`
##     shape     scale
##  1.670491 11.527173
```

```r
# Loglikelihood: with and without parameter penalty
fitted.model.new$ll
```

```
## [1] -73340.3
```

```r
fitted.model.new$ll.np
```

```
## [1] -73373.7
```

```r
# AIC
fitted.model.new$AIC
```

```
## [1] 146777.4
```

```r
# BIC
fitted.model.new$BIC
```

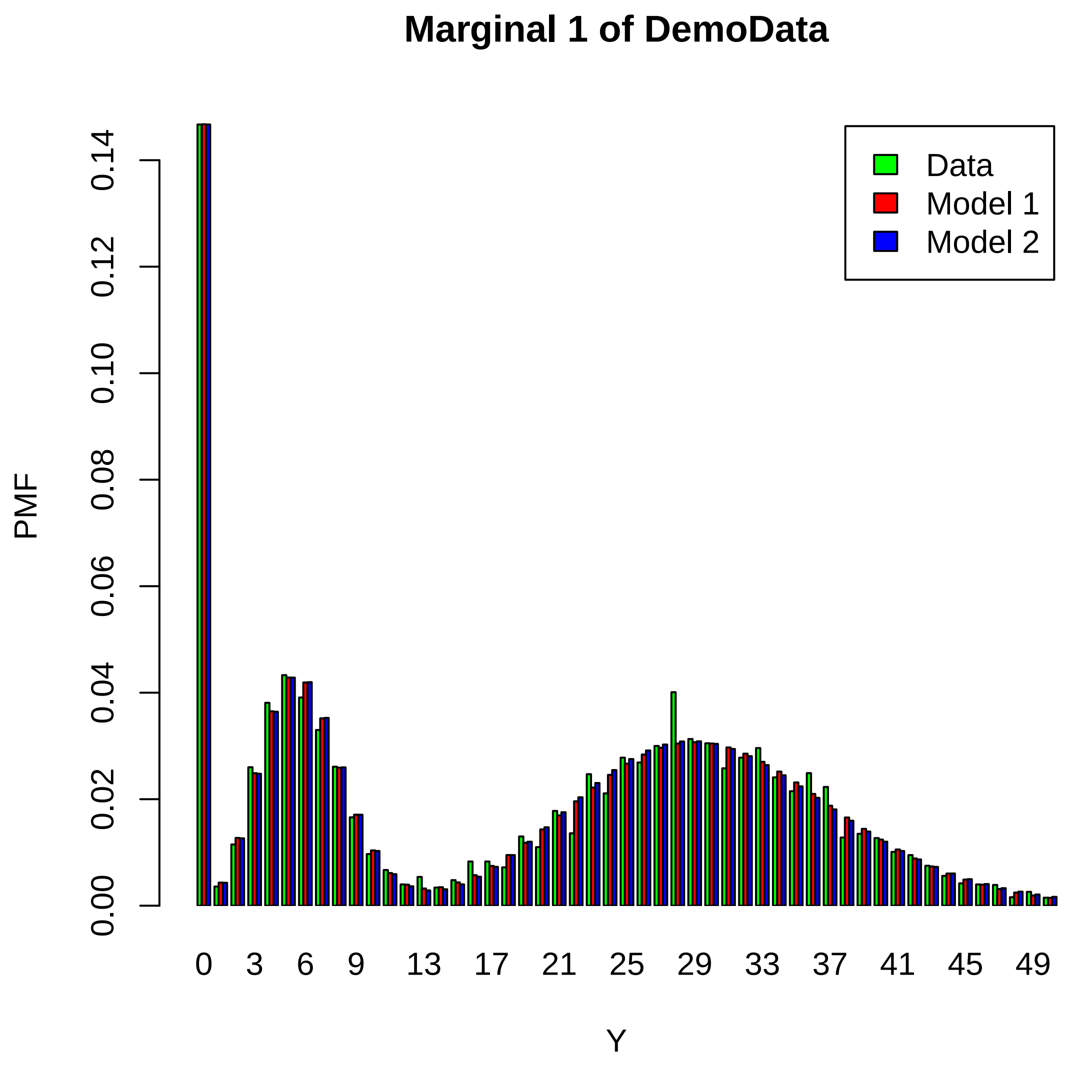
```
## [1] 146885.3
```

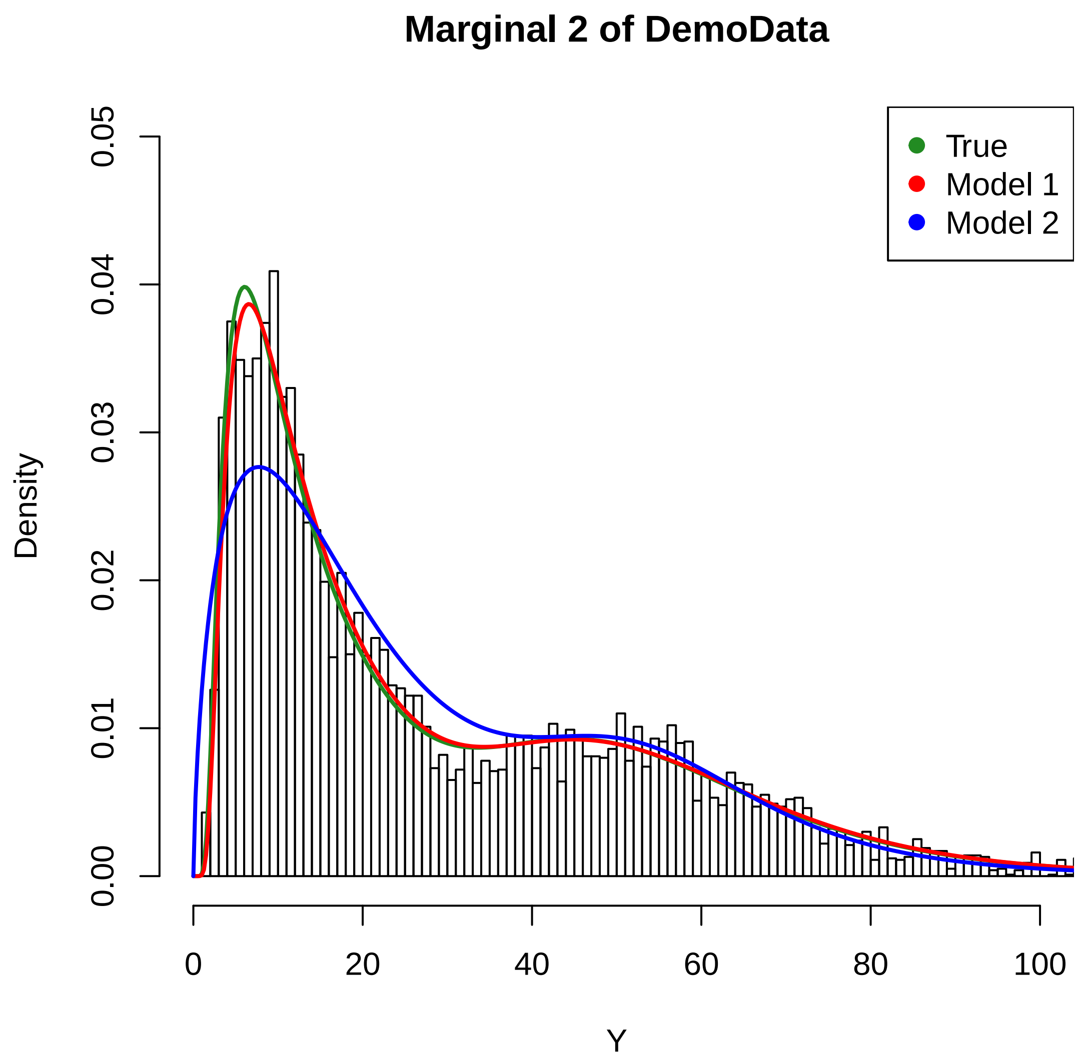
Some Model Visualization

While loglikelihood can indicate to some extent the goodness-of-fit of the two models above, one may wish to visualize the models.

Model visualization in the presence of data censoring and truncation is somewhat delicate: for example, when plotting a histogram or QQ-plot, how does one represent censored data? One solution is to simply discard those censored data points when plotting. Since we have the complete dataset at hand, we will make use of it and plot all 10,000 data points, although the model is only fitted based on a (slightly smaller) subset.

The fitted distribution by dimension against similated dataset are shown as follows.

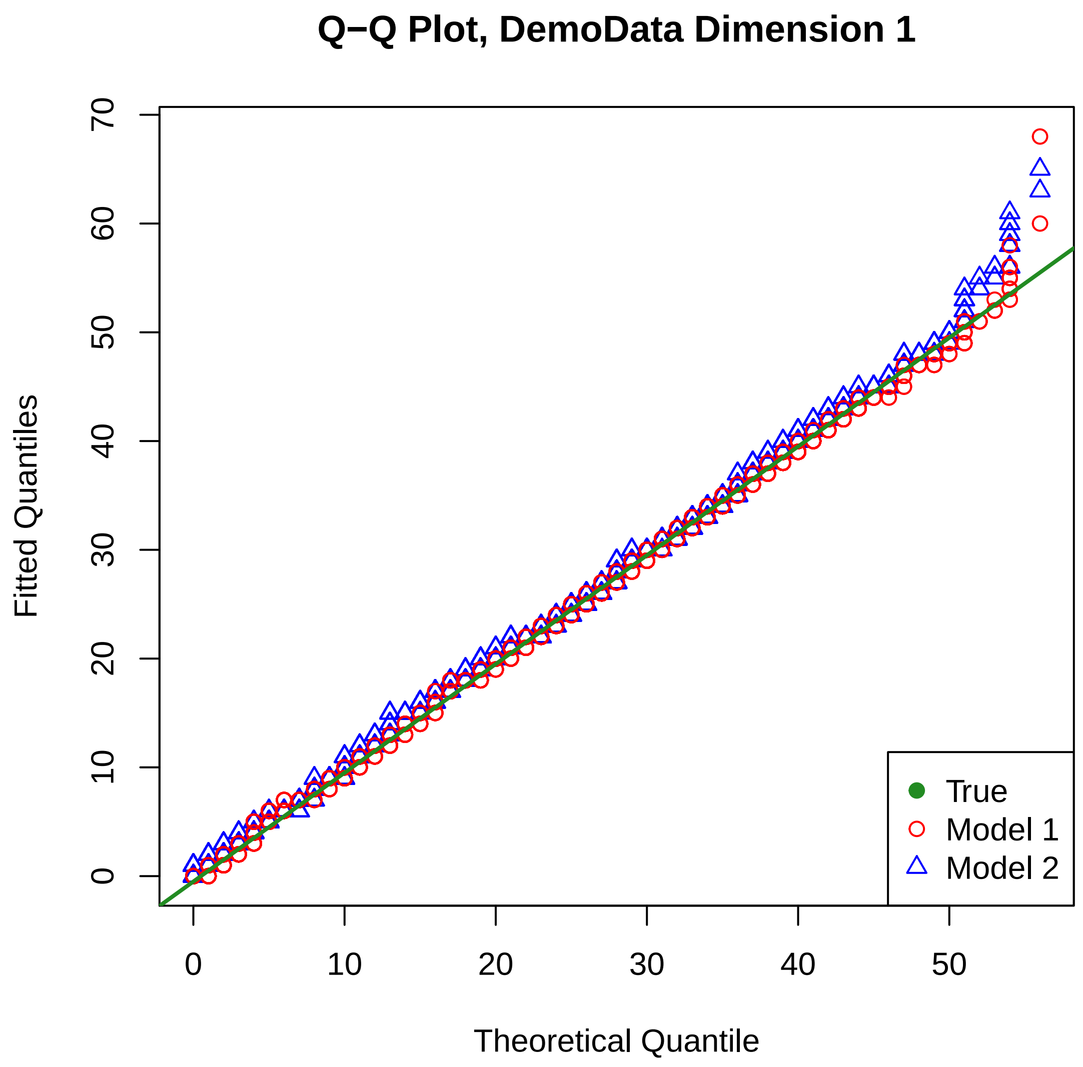

Marginal 1 of DemoData
PMF
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
0
3
6
9
13
17
21
25
29
33
37
41
45
49
Y
🟩 Data
🟥 Model 1
🟦 Model 2



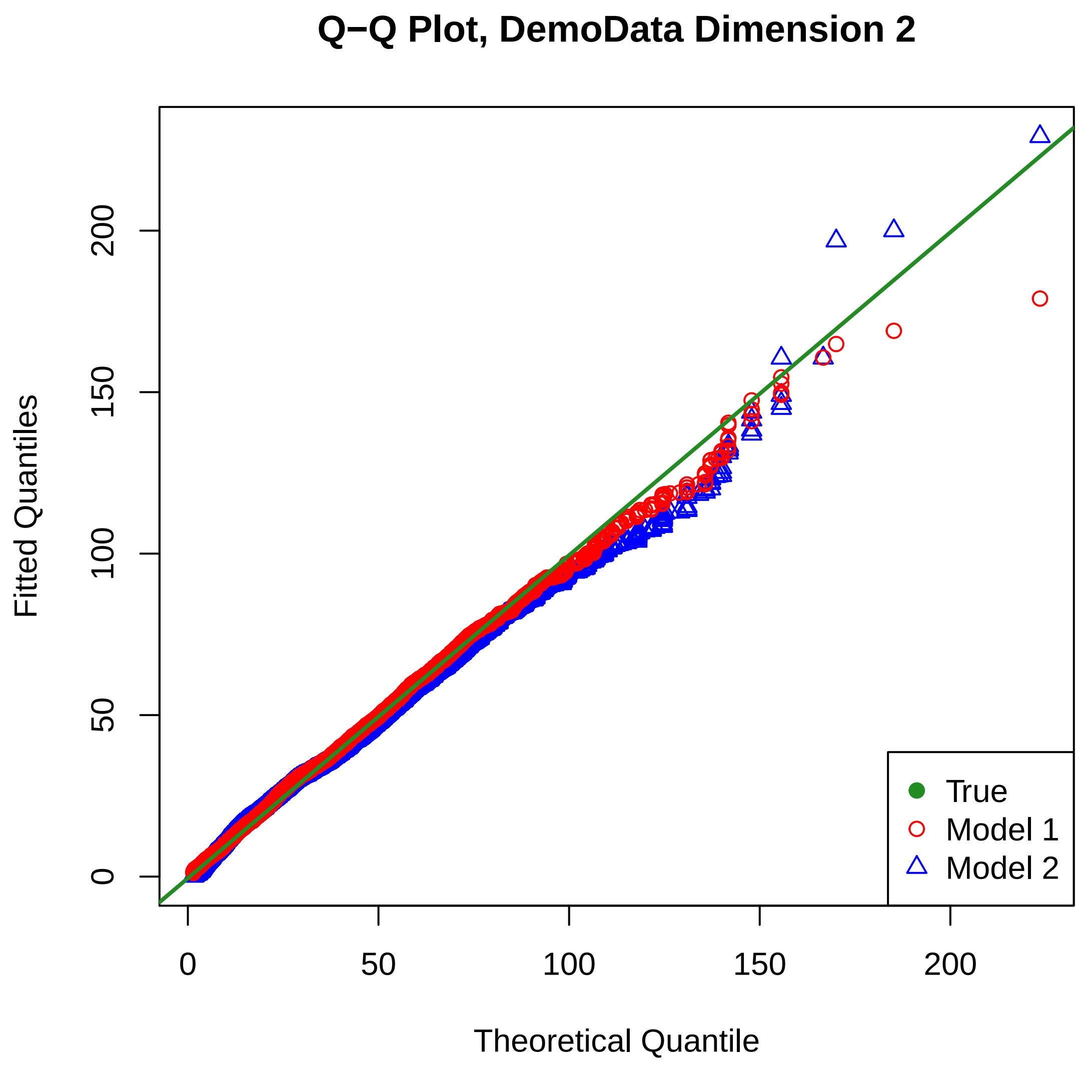
We can also obtain the QQ plot of the fitted models against the true model. For the fitted models, the distribution of the response variable for the population is a mixture of n policyholder's individual distribution with weight $1/n$ assigned to each.

It is straightforward to calculate the exact cumulative distribution function (CDF) of the fitted distribution for the population at selected points, but it may be quite computationally intensive and unstable to invert it to obtain the quantiles when the sample size is large.

For simplicity, we simulate from the fitted distribution and make the plots. Note that different random seeds may produce slightly different plots, and a larger simulation size may make the plots more stable.



Q−Q Plot, DemoData Dimension 1
Fitted Quantiles
Theoretical Quantile
0
10
20
30
40
50
60
70
● True
○ Model 1
△ Model 2



Q−Q Plot, DemoData Dimension 2
Fitted Quantiles
Theoretical Quantile
0
50
100
150
200
True
Model 1
Model 2