LRMoE: An R package for flexible actuarial loss modelling using mixture of experts regression model

Spark C. Tseung* **Andrei L. Badescu*** **Tsz Chai Fung*** **X. Sheldon Lin***

Abstract

This paper introduces a new R package, **LRMoE**, a statistical software tailor-made for actuarial applications which allows actuarial researchers and practitioners to model and analyze insurance loss frequencies and severities using the Logit-weighted Reduced Mixture-of-Experts (LRMoE) model. **LRMoE** offers several new distinctive features which are motivated by various actuarial applications and mostly cannot be achieved using existing packages for mixture models. Key features include a wider coverage on frequency and severity distributions and their zero inflation, the flexibility to vary classes of distributions across components, parameter estimation under data censoring and truncation, and a collection of insurance rate making and reserving functions. The package also provides several model evaluation and visualization functions to help users easily analyze the performance of the fitted model and interpret the model in insurance contexts.

Keywords: Multivariate Regression Analysis, Censoring and Truncation, Expectation-Conditional-Maximization Algorithm, Insurance Ratemaking and Reserving, R.

1. Introduction

The Logit-weighted Reduced Mixture-of-Experts (LRMoE) is a flexible regression model introduced by Fung *et al.* (2019b), which is regarded as the regression version of finite mixture model with the mixing weights (called the *gating function*) which depend on the covariates. We may interpret the LRMoE as clustering policyholders into different subgroup with varying probabilities. Conditioned on the subgroup component to which each policyholder is assigned, the distributional properties of loss frequency or severity are governed by mixture component functions (called the *expert function*). Model flexibility, parsimony and mathematical tractability are justified (see Fung *et al.* 2019b), demonstrating a sound theoretical foundation of LRMoE in a general insurance loss modelling perspective. Considering some specific choices of expert functions, Fung *et al.* (2019a) and Fung *et al.* (2020b) construct ECM algorithms for efficient frequency and severity model calibrations and show potential usefulness of LRMoE in terms of insurance ratemaking and reserving.

While the existing R package **flexmix** (Leisch 2004 and Grün and Leisch 2008) may perform parameter estimation for some special cases of LRMoE, it offers only limited choices of component functions (Poisson, Gaussian, Gamma and Binomial) for model fitting. Miljkovic and Grün (2016) have used its extensibility feature to prototype new mixture models with

*Address: Department of Statistical Sciences, University of Toronto, 100 St George Street, Toronto, ON M5S 3G3, Canada. Email: spark.tseung@mail.utoronto.ca (S.C. Tseung); badescu@utstat.toronto.edu (A.L. Badescu); tszchai.fung@mail.utoronto.ca (T.C. Fung); sheldon@utstat.utoronto.ca (X.S. Lin).

alternative component functions (such as Lognormal, Weibull and Burr), but users are still constrained to choosing a single parametric distribution for all the components.

This paper introduces a new R package, **LRMoE**, a statistical software tailor-made for actuarial applications which allows actuarial researchers and practitioners to model and analyze insurance loss frequencies and severities using the LRMoE model. The package offers several new distinctive features which are motivated by various actuarial applications and mostly cannot be achieved by existing packages, including:

- Wider coverage on frequency and severity distributions: Apart from the severity distributions covered by Miljkovic and Grün (2016), the package also covers more frequency expert functions important for actuarial loss modelling, including negative binomial distribution and gamma-count distribution.

- Zero-inflated distributions: Often actuaries are more interested in analyzing the aggregate loss for each policyholder instead of considering frequency and severity separately. In this situation, it is common in practice to observe excessive zeroes, which motivates the use of zero-inflated expert functions in the LRMoE, which is offered in this package. Note that efficient computation of zero-inflated LRMoE requires defining an additional latent variable (see Section 2.2 for details), further hindering the effectiveness of using the extensibility feature of **flexmix**.

- Varying classes of distributions across components: Insurance loss data may exhibit a mismatch between body and tail behaviors, which should be captured using different distributions. One approach is to choose two distributions and combine them using a peaks-over-threshold method (see e.g. Lee *et al.* 2012 and Scollnik and Sun 2012). Another is to consider a finite mixture model based on different component distributions (see e.g. Blostein and Miljkovic 2019). The **LRMoE** package is similar to the latter, and users can select different expert functions across different mixture components, which allows for more flexible and realistic modelling of data.

- Incomplete data: In many actuarial applications, including reinsurance, operational risk management, deductible ratemaking and loss reserving, censored and truncated data are often observed and need to be dealt with. Censoring and truncation of LRMoE is introduced by Fung *et al.* (2020a) with the expert functions restricted to univariate gamma distribution. The new package removes such restriction by offering users versatility to fit randomly censored and truncated multivariate data with many choices of expert functions.

- Model selection and visualization: In addition to model fitting function, the new package also provides several model evaluation (AIC, BIC) and visualization (e.g. latent class probabilities, covariate influence) functions to help users easily analyze the performance of the fitted model and interpret the fitted model in the insurance context.

- Insurance ratemaking and reserve calculation: The package further contains a number of pricing and reserving functions (e.g. mean, variance, value-at-risk, conditional tail expectation), which enable actuaries to simultaneously perform ratemaking to multiple insurance contracts with different characteristics, based on abundant choices of premium principles.

The paper is organized as follows. Section 2 reviews the LRMoE model and parameter estimation using the ECM algorithm. In Section 3, we use a simulated dateset to demonstrate the basic fitting procedure in the **LRMoE** package. Section 4 contains more package utilities such as parameter initialization, model visualization and pricing function, which are illustrated using a French auto insurance dataset. The paper is concluded with some remarks in Section 5. The source code, package documendation and replication code for all examples in this paper are available on https://github.com/sparktseung/LRMoE.

2. LRMoE Model and Parameter Estimation

In this section, we provide a brief overview of the LRMoE model proposed in Fung *et al.* (2019b), and discuss the ECM algorithm for paramater estimation. For brevity of presentation, we will assume, in Sections 2.1 and 2.2, that all response variables (claim frequency or severity) are observed exactly. In section 2.3, we will address data truncation and censoring for the LRMoE model.

2.1. Logit-Weighted Reduced Mixture of Experts

Let $\boldsymbol{x}_i = (x_{i0}, x_{i1}, \ldots, x_{iP})^T$ denote the $(P+1)$-dimensional covariate vector for policyholder i $(i = 1, 2, \ldots, n)$ with intercept $x_{i0} = 1$, and $\boldsymbol{y}_i = (y_{i1}, y_{i2}, \ldots, y_{iD})^T$ denote the D-dimensional vector of their response variables, which can be either claim frequency or severity. Let $\boldsymbol{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n)^T$ and $\boldsymbol{Y} = (\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_n)^T$ denote all covarites and responses for a group of n policyholders.

Based on the covariates, policyholder i is classified into one of g latent risk classes by a logit gating function

$$\pi_j(\boldsymbol{x}_i; \boldsymbol{\alpha}_j) = \frac{exp(\boldsymbol{\alpha}_j^T \boldsymbol{x}_i)}{\sum_{j'=1}^{g} exp(\boldsymbol{\alpha}_{j'}^T \boldsymbol{x}_i)}, \quad j = 1, 2, \ldots, g, \tag{1}$$

where $\boldsymbol{\alpha}_j = (\alpha_{j0}, \alpha_{j1}, \ldots, \alpha_{jP})^T$ is a vector of regression coefficients for latent class j.

Given the assignment of latent class j, the response variables $\boldsymbol{y}_i = (y_{i1}, y_{i2}, \ldots, y_{iD})^T$ are conditionally independent, with the d-th dimension probability density function (or probability mass function) given by

$$g_{jd}(y_{id}; \delta_{jd}, \boldsymbol{\psi}_{jd}) = \delta_{jd} \mathbf{1}\{y_{id} = 0\} + (1 - \delta_{jd}) f_{jd}(y_{id}; \boldsymbol{\psi}_{jd}) \tag{2}$$

where δ_{jd} is a probability mass at zero, $\mathbf{1}\{y_{id} = 0\}$ is the indicator function, and $f_{jd}(y_{id}; \boldsymbol{\psi}_{jd})$ is the density of some commonly used distribution with parameters $\boldsymbol{\psi}_{jd}$ for modelling insurance losses. Table 1 gives a list of parametric distributions supported by the **LRMoE** package. Note that a probability mass δ_{jd} at zero allows for more realistic modelling of insurance data which usually exhibit excess zeros. As a naming convention, we will refer to g_{jd} as a *component distribution/expert function*, and f_{jd} as its *positive part*, although claim frequency distributions (e.g. Poisson) also have some probability mass at zero.

Under LRMoE, the conditional probability density function (or probability mass fuction) of $\boldsymbol{y}_i$ given covariates $\boldsymbol{x}_i$ is

$$h(\boldsymbol{y}_i; \boldsymbol{x}_i, \boldsymbol{\alpha}, \boldsymbol{\delta}, \boldsymbol{\Psi}) = \sum_{j=1}^{g} \pi_j(\boldsymbol{x}_i; \boldsymbol{\alpha}_j) \times \prod_{d=1}^{D} g_{jd}(y_{id}; \delta_{jd}, \boldsymbol{\psi}_{jd}) \tag{3}$$

Table 1: Distributions Supported by **LRMoE**

Root	Distribution	$f_{jd}(y)$	Parameters
`gamma`	Gamma	$\frac{1}{\theta^m\Gamma(m)}y^{m-1}e^{-y/\theta}$	$m>0$, $\theta>0$
`lnorm`	Lognormal	$\frac{1}{y\sigma\sqrt{2\pi}}\exp\left[-\frac{(\log y-\mu)^2}{2\sigma^2}\right]$	$\mu\in\mathbb{R}$, $\sigma>0$
`invgauss`	Inverse Gaussian	$\sqrt{\frac{\lambda}{2\pi y^3}}\exp\left[-\frac{\lambda(y-\mu)^2}{2\mu^2 y}\right]$	$\mu>0$, $\lambda>0$
`weibull`	Weibull	$\frac{k}{\lambda}\left(\frac{y}{\lambda}\right)^{k-1}\exp\left[-\left(\frac{y}{\lambda}\right)^k\right]$	$k>0$, $\lambda>0$
`burr`	Burr	$\frac{ck}{\lambda}\left(\frac{y}{\lambda}\right)^{c-1}\left[1+\left(\frac{y}{\lambda}\right)^c\right]^{-k-1}$	$k>0$, $c>0$, $\lambda>0$
`poisson`	Poisson	$e^{-\lambda}\frac{\lambda^y}{y!}$	$\lambda>0$
`nbinom`	Negative Binomial	$\binom{y+n-1}{n-1}p^n(1-p)^y$	$n\in\mathbb{N}^+$, $0<p<1$
`gammacount`	Gamma-Count	`pgamma(ms, ys, 1)` `- pgamma(ms, (y+1)s, 1)`	$m>0$, $s>0$
`ZI-root`	All above	$g_{jd}=\delta_{jd}\mathbf{1}\{y=0\}+(1-\delta_{jd})f_{jd}$	$0<\delta_{jd}<1$

where $\boldsymbol{\alpha}=(\boldsymbol{\alpha}_1,\boldsymbol{\alpha}_2,\ldots,\boldsymbol{\alpha}_g)^T$ is a $g\times(P+1)$-matrix of mixing weights with $\boldsymbol{\alpha}_g=(0,0,\ldots,0)^T$ representing the default class, $\boldsymbol{\delta}=(\delta_{jd})_{1\le j\le g,1\le d\le D}$ is a $(g\times D)$-matrix of probability masses at zero by component and by dimension, and $\boldsymbol{\Psi}=\{\boldsymbol{\psi}_{jd}:1\le j\le g,1\le d\le D\}$ is a list of parameters for the positive part f_{jd} by component and by dimension. Note that $\boldsymbol{\alpha}_g=(0,0,\ldots,0)^T$ ensures that the model is identifiable, i.e. there is a one-to-one mapping between the regression distributions and the parameters (see Jiang and Tanner (1999) and Fung *et al.* (2019a)).

For a group of n policyholders, the likelihood function is given by

$$L(\boldsymbol{\alpha},\boldsymbol{\delta},\boldsymbol{\Psi};\boldsymbol{X},\boldsymbol{Y})=\prod_{i=1}^{n}h(\boldsymbol{y}_i;\boldsymbol{x}_i,\boldsymbol{\alpha},\boldsymbol{\delta},\boldsymbol{\Psi})=\prod_{i=1}^{n}\left\{\sum_{j=1}^{g}\pi_j(\boldsymbol{x}_i;\boldsymbol{\alpha}_j)\times\prod_{d=1}^{D}g_{jd}(y_{id};\delta_{jd},\boldsymbol{\psi}_{jd})\right\}. \quad (4)$$

2.2. Parameter Estimation

For parameter estimation in finite mixture models, the Expectation-Maximization (EM) algorithm is commonly used (see e.g. Dempster *et al.* (1977) and McLachlan and Peel (2004)). However, for the LRMoE model, the M-step requires maxmization of a non-concave function over all elements of $\boldsymbol{\alpha}$. Fung *et al.* (2019a) thus use the Expectation-Conditional-Maximization (ECM) algorithm (Meng and Rubin (1993)) which breaks the M-step into several substeps. The ECM algorithm implemented in the **LRMoE** package is described as follows.

Denote $\boldsymbol{\Phi}=(\boldsymbol{\alpha},\boldsymbol{\delta},\boldsymbol{\Psi})$ as the parameters to estimate. For $i=1,2,\ldots,n$, we introduce a latent random vector $\boldsymbol{Z}_i=(Z_{i1},Z_{i2},\ldots,Z_{ig})^T$, where $Z_{ij}=1$ if $\boldsymbol{y}_i$ comes from the j-

th component distribution and $Z_{ij} = 0$ otherwise. For $d = 1, 2, \ldots, D$, we further write $Z_{ij} = Z_{ijd} = Z_{ijd0} + Z_{ijd1}$, where $Z_{ijd1} = 1$ if the d-th dimension of $\boldsymbol{y}_i$ comes from the positive part f_{jd} of the j-th component and $Z_{ijd1} = 0$ otherwise.

The complete-data loglikelihood function is given by

$$\begin{aligned} l^{\text{com}}(\boldsymbol{\Phi}; \boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}) &= \sum_{i=1}^{n}\sum_{j=1}^{g} Z_{ij}\left\{\log \pi_j(\boldsymbol{x}_i; \boldsymbol{\alpha}_j) + \sum_{d=1}^{D} \log g_{jd}(y_{id}; \delta_{jd}, \boldsymbol{\psi}_{jd})\right\} \\ &= \sum_{i=1}^{n}\sum_{j=1}^{g} Z_{ij} \log \pi_j(\boldsymbol{x}_i; \boldsymbol{\alpha}_j) + \sum_{i=1}^{n}\sum_{j=1}^{g}\sum_{d=1}^{D} \{Z_{ijd0} \log \delta_{jd} + Z_{ijd1} \log(1 - \delta_{jd})\} \\ &\quad + \sum_{i=1}^{n}\sum_{j=1}^{g}\sum_{d=1}^{D} Z_{ijd1} \log f_{jd}(y_{id}; \boldsymbol{\psi}_{jd}). \end{aligned} \tag{5}$$

E-Step:

For each $i = 1, 2, \ldots, n$, the random vector $\boldsymbol{Z}_i$ follows a multinomial distribution with count 1 and probabilities $(\pi_1(\boldsymbol{x}_i; \boldsymbol{\alpha}_1), \pi_2(\boldsymbol{x}_i; \boldsymbol{\alpha}_2), \ldots, \pi_g(\boldsymbol{x}_i; \boldsymbol{\alpha}_g))$. Given $Z_{ij} = 1$, the conditional distribution of Z_{ijd0} is Bernoulli with probability δ_{jd}. Hence, at the t-th iteration, the posterior expectations of Z_{ij}, Z_{ijd0} and Z_{ijd1} are

$$\begin{aligned} z_{ij}^{(t)} &= \mathsf{E}\{Z_{ij} | \boldsymbol{\Phi}^{(t-1)}, \boldsymbol{X}, \boldsymbol{Y}\} = \mathsf{P}\{Z_{ij} = 1 | \boldsymbol{\Phi}^{(t-1)}, \boldsymbol{X}, \boldsymbol{Y}\} \\ &= \frac{\pi_j(\boldsymbol{x}_i; \boldsymbol{\alpha}_j^{(t-1)}) \times \prod_{d=1}^{D} g_{jd}(y_{id}; \delta_{jd}^{(t-1)}, \boldsymbol{\psi}_{jd}^{(t-1)})}{\sum_{j'=1}^{g} \pi_{j'}(\boldsymbol{x}_i; \boldsymbol{\alpha}_{j'}^{(t-1)}) \times \prod_{d=1}^{D} g_{j'd}(y_{id}; \delta_{j'd}^{(t-1)}, \boldsymbol{\psi}_{j'd}^{(t-1)})}, \end{aligned} \tag{6}$$

$$\begin{aligned} z_{ijd0}^{(t)} &= \mathsf{E}\{Z_{ijd0} | \boldsymbol{\Phi}^{(t-1)}, \boldsymbol{X}, \boldsymbol{Y}\} \\ &= \mathsf{P}\{Z_{ijd0} = 1 | \boldsymbol{\Phi}^{(t-1)}, \boldsymbol{X}, \boldsymbol{Y}, Z_{ij} = 1\} \times \mathsf{P}\{Z_{ij} = 1 | \boldsymbol{\Phi}^{(t-1)}, \boldsymbol{X}, \boldsymbol{Y}\} \\ &= \frac{\delta_{jd}^{(t-1)} \mathbf{1}\{y_{id} = 0\}}{\delta_{jd}^{(t-1)} \mathbf{1}\{y_{id} = 0\} + (1 - \delta_{jd}^{(t-1)}) F_{jd}(0; \boldsymbol{\psi}_{jd}^{(t-1)})} \times z_{ij}^{(t)}, \end{aligned} \tag{7}$$

and

$$z_{ijd1}^{(t)} = z_{ij}^{(t)} - z_{ijd0}^{(t)} \tag{8}$$

where F_{jd} is the cumulative distribution function of the positive part f_{jd}.

CM-Step:

In the CM-step, we aim to maximize $Q(\boldsymbol{\Phi}; \boldsymbol{\Phi}^{(t-1)}, \boldsymbol{X}, \boldsymbol{Y})$, which can be decomposed into three parts as

$$Q(\boldsymbol{\Phi}; \boldsymbol{\Phi}^{(t-1)}, \boldsymbol{X}, \boldsymbol{Y}) = Q_{\boldsymbol{\alpha}}^{(t)} + Q_{\boldsymbol{\delta}}^{(t)} + Q_{\boldsymbol{\Psi}}^{(t)} \tag{9}$$

where

$$Q_{\boldsymbol{\alpha}}^{(t)} = \sum_{i=1}^{n}\sum_{j=1}^{g} z_{ij}^{(t)} \log \pi_j(\boldsymbol{x}_i; \boldsymbol{\alpha}_j), \tag{10}$$

$$Q_{\boldsymbol{\delta}}^{(t)} = \sum_{i=1}^{n}\sum_{j=1}^{g}\sum_{d=1}^{D}\left\{z_{ijd0}^{(t)}\log\delta_{jd} + z_{ijd1}^{(t)}\log(1-\delta_{jd})\right\}, \tag{11}$$

and

$$Q_{\boldsymbol{\Psi}}^{(t)} = \sum_{i=1}^{n}\sum_{j=1}^{g}\sum_{d=1}^{D} z_{ijd1}^{(t)}\log f_{jd}(y_{id};\boldsymbol{\psi}_{jd}). \tag{12}$$

To maximize $Q_{\boldsymbol{\alpha}}^{(t)}$, we use the same conditional maximization as described in Fung *et al.* (2019a). We first maximize it with respect to $\boldsymbol{\alpha}_1$ with $\boldsymbol{\alpha}_j$ fixed at $\boldsymbol{\alpha}_j^{(t-1)}$ for $j = 2, 3, \ldots, g-1$. The next step is to maximize with respect to $\boldsymbol{\alpha}_2$ with updated $\boldsymbol{\alpha}_1^{(t)}$ and other $\boldsymbol{\alpha}_j$ fixed at $\boldsymbol{\alpha}_j^{(t-1)}$ for $j = 3, 4, \ldots, g-1$. The process continues until all $\boldsymbol{\alpha}$'s have been updated. For obtaining each $\boldsymbol{\alpha}_j^{(t)}$, the Iteratively Reweighted Least Square (IRLS) approach (Jordan and Jacobs 1994) is used until convergence.

For $Q_{\boldsymbol{\delta}}^{(t)}$, each δ_{jd} can be updated using the following closed-form solution

$$\delta_{jd}^{(t)} = \frac{\sum_{i=1}^{n} z_{ijd0}^{(t)}}{\sum_{i=1}^{n}(z_{ijd0}^{(t)} + z_{ijd1}^{(t)})}. \tag{13}$$

The maximization of $Q_{\boldsymbol{\Psi}}^{(t)}$ can also be divided into smaller problems by component and by dimension. For updating each $\boldsymbol{\psi}_{jd}^{(t)}$, closed-form solutions are only available for very special distributions (e.g. Poisson, Lognormal). Numerical optimization is used in most cases, especially when the observation $\boldsymbol{y}_i$'s are not observed exactly (see Section 2.3).

As discussed in McLachlan and Peel (2004), mixture of severity distributions may have unbounded likelihood, which leads to spurious models with extremely large or small parameter values. In the **LRMoE** package, we adopt the same maximum a posteriori (MAP) approach in Fung *et al.* (2020b), which uses appropriate prior distributions to penalize the magnitude of fitted parameters (see Section 3).

2.3. LRMoE with Censoring and Truncation

Censoring and truncation are common in insurance data sets and need to be dealt with. For example, when a policy limit is applied, loss amounts above the limit will be recorded as the limit only, which creates right censoring of the complete loss data; when a policy deductible is applied, loss amounts below the deductible are not reported to the insurer, thus leading to left truncation.

Fung *et al.* (2020a) have discussed the LRMoE model with censoring and truncation, where all component distributions are Gamma. For parameter estimation with data censoring and truncation, the ECM algorithm in section 2.2 is slightly modified, with an additional E-step to remove the uncertainties arising from censoring and truncation. Since the main purpose of this paper is to demonstrate the application of **LRMoE** package, we will omit the details and refer interested readers to the cited paper.

For all distributions included in it, the **LRMoE** package can perform parameter estimation in the presence of data truncation and censoring. Consequently, the user's input is slightly

different from existing packages for mixture models. A detailed example on model fitting in our package is given in Section 3.

2.4. Ratemaking and Reserving in LRMoE

The model structure of LRMoE allows for easy computation of quantities relevant to actuarial ratemaking and reserving (see Fung *et al.* 2019b). At the policyholder level, the moments and common measures of dependence (e.g. Kendall's tau and Spearman's rho) of $\boldsymbol{y}_i$ can be computed in simple forms. The value-at-risk (VaR) and conditional tail expectation (CTE) can also be numerically solved without much difficulty. Various premium principles can be applied to price insurance contracts, including pure premium, standard deviation (SD) premium, limited expected value (LEV) and stop-loss (SL) premium. Risk measures can also be calculated for each individual policyholder (e.g. 99%-VaR). At the portfolio level, simulation can be conducted to obtain the distribution of the aggregate loss of all policyholders, which is useful for calculating the total loss reserve and premium calculation. The simulation process is facilitated by a data simulator included in our package (see Section 4.5).

3. Example: Simulated Dataset

In this section, we will demonstrate how to fit an LRMoE model using a simulated dataset `DemoData` included in the package. The variables in `DemoData` are described in Table 2, and the dataset is generated by an LRMoE model given in Table 3. It can be loaded with the following lines.

```
# The package and source code are available on github.
# The installation requires two lines of code.
> library(devtools)
> install_github("sparktseung/LRMoE")

# Load DemoData which contains four matrices
# X, Y are complete datasets with 10,000 rows
# X.obs, Y.obs are subject to data truncation and censoring
> library(LRMoE)
> data(DemoData)
```

To address data truncation and censoring, the user's input of response $\boldsymbol{Y}$ is different from existing packages. For each dimension d of observation $\boldsymbol{y}_i$, instead of a single numeric input, a quadruple $0 \leq t_{id}^l \leq y_{id}^l \leq y_{id}^u \leq t_{id}^u \leq \infty$ is needed, where t_{id}^l and t_{id}^u are the lower and upper bounds of truncation, and y_{id}^l and y_{id}^u are the lower and upper bounds of censoring. The exact value of y_{id} lies between censoring bounds such that $y_{id}^l \leq y_{id} \leq y_{id}^u$. For a sample of size n, an $n \times (4D)$-matrix is needed, where each $n \times 4$-block describes one dimension of $\boldsymbol{Y}$. Sample rows of `Y.obs` in `DemoData` are shown as follows.

```
# Complete data: no truncation, no censoring
> Y.obs[1,]
```

tl.1	yl.1	yu.1	tu.1	tl.2	yl.2	yu.2	tu.2
0.0000	5.0000	5.0000	Inf	0.0000	40.2224	40.2224	Inf

Table 2: Description of `DemoData`

Covariate[1]	Name	Description
x_{i0}	`intercept`	Constant 1
x_{i1}	`sex`	1 for male, 0 for female
x_{i2}	`aged`	Driver's Age: 20–80
x_{i3}	`agec`	Car's Age: 0–10
x_{i4}	`region`	1 for urban, 0 for rural
Response	**Name**	**Description**
y_{i1}	`Y[,1]`	Claim count from business line 1
y_{i2}	`Y[,2]`	Claim severity from business line 2
Row Index	**`Y[,1]`**	**`Y[,2]`**
1–6000	No censoring or truncation	No censoring or truncation
6001–8000	No censoring or truncation	Left-truncated at 5 [2]
8001–10000	No censoring or truncation	Right-censored at 100

[1] All covariates are generated independently and uniformly at random.
[2] The complete dataset `(X, Y)` contains 10,000 rows. As a result of left-truncating `Y[,2]`, 172 rows of data are discarded, and the observed dataset `(X.obs, Y.obs)` has 9828 rows only.

```
# Y[,2] is left-truncated at 5
> Y.obs[6002,]

    tl.1     yl.1     yu.1     tu.1     tl.2     yl.2     yu.2     tu.2
  0.0000   0.0000   0.0000      Inf   5.0000  81.3488  81.3488      Inf

# Y[,2] is right-censored at 100
> Y.obs[7884,]

    tl.1     yl.1     yu.1     tu.1     tl.2     yl.2     yu.2     tu.2
       0        7        7      Inf        0      100      Inf      Inf
```

To fit an LRMoE model, the user needs to specify the following inputs: number of latent classes (`n.comp`), what component distributions to use (`comp.dist`) for each dimension and each component, initial guess of model parameters (`alpha.init` for logit regression coefficients, `zero.init` for zero-inflation probabilities and `params.init` for parameters of the positive part of component distributions).

For illustration purposes, we first assume that the user's choice of component distributions coincides with the true model, and the initial guesses of parameters are also close to the true ones. As shown in the following sample code of initialization, the user has chosen a two-component mixture. For the first dimension of response variable, the user specifies the second latent component `comp.dist[1,2]` to be a zero-inflated gamma-count distribution, where the initial guess of zero-inflation `zero.init[1,2]` is 0.5 and the gamma-count parameters `params.init[[1]][[2]]` are $(40, 0.8)$.

```
# Number of latent classes (component distributions)
> n.comp = 2  # = g
```

Table 3: True Model of `DemoData`

Logit Regression Coefficients:

$$\boldsymbol{\alpha} = \begin{bmatrix} -0.50 & 1.00 & -0.05 & 0.10 & 1.25 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Component Distributions:

	$j=1$			$j=2$		
	`comp.dist`	δ_{jd}	$\boldsymbol{\psi}_{jd}$	`comp.dist`	δ_{jd}	$\boldsymbol{\psi}_{jd}$
$d=1$	`poisson`	0	$\lambda = 6$	`ZI-gammacount`	0.20	$m = 30, s = 0.50$
$d=2$	`lnorm`	0	$\mu = 4, \sigma = 0.30$	`invgauss`	0	$\mu = 20, \lambda = 20$

Table 4: Fitted Model 1 of `DemoData`

Logit Regression Coefficients:

$$\hat{\boldsymbol{\alpha}} = \begin{bmatrix} -0.3927 & 0.9837 & -0.0493 & 0.0910 & 1.1527 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Component Distributions:

	$j=1$			$j=2$		
	`comp.dist`	$\hat{\delta}_{jd}$	$\hat{\boldsymbol{\psi}}_{jd}$	`comp.dist`	$\hat{\delta}_{jd}$	$\hat{\boldsymbol{\psi}}_{jd}$
$d=1$	`poisson`	0	$\lambda = 5.77$	`ZI-gammacount`	0.19	$m = 29.52, s = 0.49$
$d=2$	`lnorm`	0	$\mu = 4.01, \sigma = 0.30$	`invgauss`	0	$\mu = 19.92, \lambda = 22.06$

Table 5: Fitted Model 2 of `DemoData`

Logit Regression Coefficients:

$$\tilde{\boldsymbol{\alpha}} = \begin{bmatrix} -0.3674 & 0.9811 & -0.0494 & 0.0917 & 1.1483 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Component Distributions:

	$j=1$			$j=2$		
	`comp.dist`	$\tilde{\delta}_{jd}$	$\tilde{\boldsymbol{\psi}}_{jd}$	`comp.dist`	$\tilde{\delta}_{jd}$	$\tilde{\boldsymbol{\psi}}_{jd}$
$d=1$	`ZI-poisson`	0.009	$\lambda = 5.87$	`ZI-nbinom`	0.195	$n = 30, p = 0.499$
$d=2$	`burr`	0	$k = 1.12, c = 5.57,$ $\lambda = 56.99$	`gamma`	0	$m = 1.67, \theta = 11.53$

```
# Number of response dimension
> dim.m = 2   # = d

# A matrix of strings to 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
# by component (row) and by covariate (column)
# Last row must be zero for default class
# Dimension is g * (P+1)
> 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
# by dimension (row) and by component (column)
# Must be zero for non-zero-inflated component distributions
> zero.init = matrix( c(0, 0.5,
                        0,   0),
                      nrow = dim.m, byrow = TRUE)

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

By default, the fitting function imposes a penalty on the magnitude of parameters to avoid spurious models with extremely large or small parameter values. This is implemented by specifying hyper parameters for the prior distributions of model parameters. For simplicity, positive parameters are given Gamma priors, real parameters are given normal priors with mean zero, while parameters with bounded ranges are not penalized at all.

For example, the following lines indicate that the prior distribution are: logit regression coefficients $\alpha_{jp} \sim \mathrm{N}(0, 5^2)$, Lognormal `meanlog` $\mu \sim \mathrm{N}(0, 1^2)$, Lognormal `sdlog` $\sigma \sim \mathrm{Gamma}(9, 0.5)$, etc. No penalty is imposed on zero-inflation probabilities.

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

# Penalty for component distribution parameters
# List structure is the similar to params.init
> hyper.params = list( list(c(9, 0.5),    c(9, 0.5, 9, 0.5)),
                       list(c(1, 9, 0.5), c(9, 0.5, 9, 0.5))
                     )
```

With all input argument properly defined, the model fitting function can be called as follows. It is optional to print intermediate parameter updates on screen by setting `print = TRUE`.

```
> fitted.model = 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, which contains model specification (`n.comp`, `comp.dist`), intializations (e.g. `alpha.init`), estimated parameters (e.g. `alpha.fit`), loglikelihood (`ll`, `ll.np` with no penalty) and information criteria of the fitted model (`AIC`, `BIC`). They can be inspected using standard R methods. Examples are given below.

```
# Check LRMoE model specification
> fitted.model$comp.dist

      comp 1    comp 2
dim 1 "poisson" "ZI-gammacount"
dim 2 "lnorm"   "invgauss"

# Inspect fitted logit regression coeficients
> fitted.model$alpha.fit

        intercept       sex   agedriver    agecar   region
comp 1 -0.3926849 0.9836549 -0.04925036 0.0910102 1.152698
comp 2  0.0000000 0.0000000  0.00000000 0.0000000 0.000000

# Inspect fitted parameters for the second dimension of Y
> fitted.model$params.fit[["dim 2"]] # or fitted.model$params.fit[[2]]

$`comp 1`
 meanlog   sdlog
     4.0     0.3

$`comp 2`
    mean    scale
19.77823 21.42792
```

The fitted model is summarized in Table 4. The parameter estimates are quite close to the true ones. Considering simulated random noises and loss of information due to censoring and truncation, the fitting function is able to identify the true model when it is known.

In practice, when the true underlying model is not known, the user needs to perform some preliminary analysis on the dataset to determine model specification and parameter initialization. Table 5 contains the parameter estimates of another user-specified LRMoE model for `DemoData`, which has quite different component distributions compared with the true model.

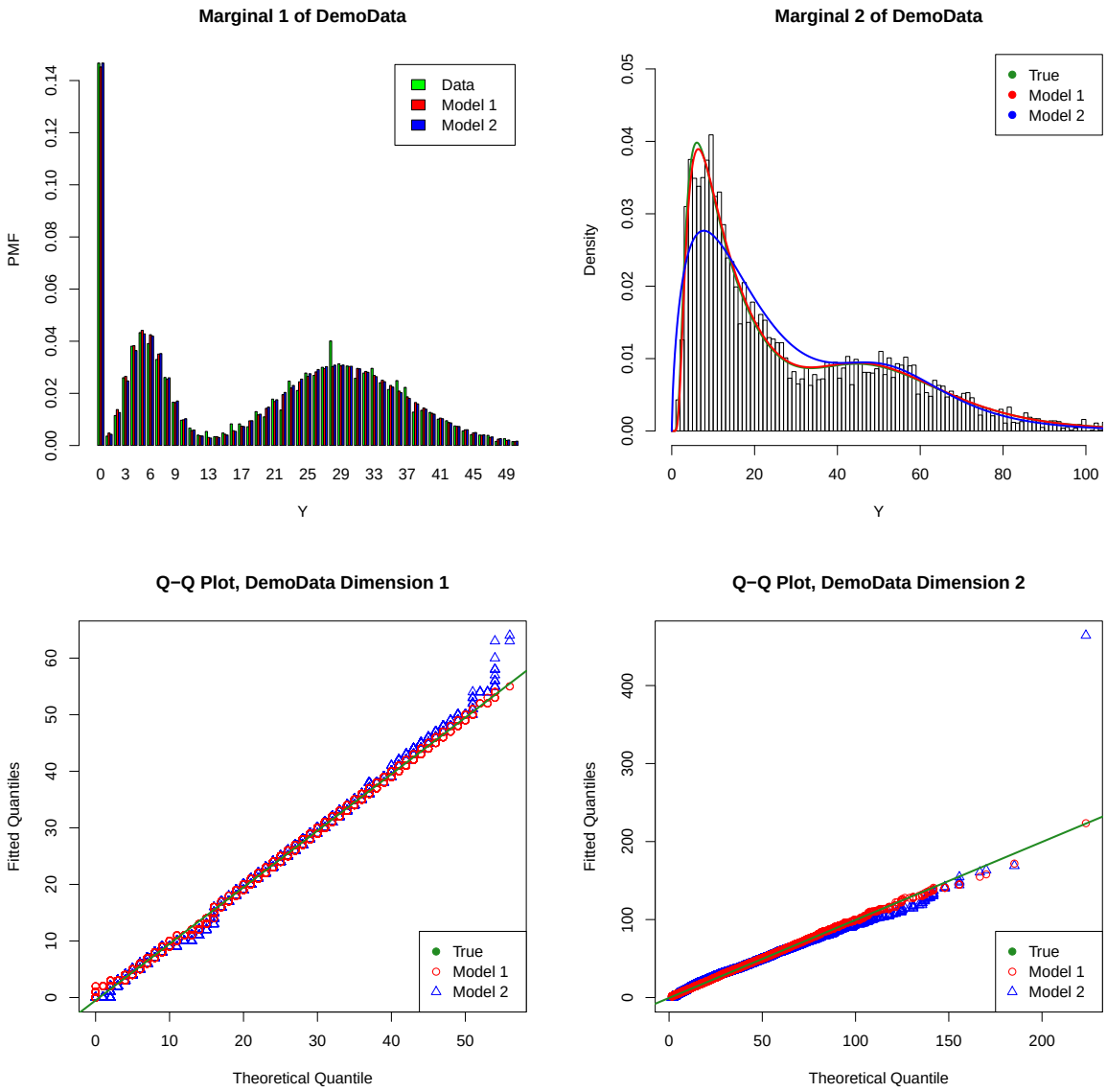

Figure 1: Fitting Results of `DemoData`

The fitted loglikelihood values for Model 1 and Model 2 are -72885.57 and -73374.48, respectively. A graphical comparison of the two models are given in Figure 1. While both models have similar fitting performance for claim frequency, Model 2 is noticeably worse in fitting small and extreme values of claim severity.

4. Example: Real Dataset

In this section, we illustrate how to fit an LRMoE model to a real insurance dataset. As the basic fitting procedure has been discussed in the previous section, we will focus on data preprocessing and other utilities of our package, including parameter initialization, computation parallelization, actuarial pricing functions and model visualization.

Covariate	Name	Description
x_{i0}	`Intercept`	Constant 1. Default class for categorical variables.
x_{i1}	`CarAge`	Vehicle age in years. Range: $0 \sim 100$
x_{i2}	`DriverAge`	Driver's age in years. Range: $18 \sim 99$
$x_{i3} \sim x_{i,13}$	`Power`	Power of the car as ordered categorical: d $\sim$ o. Default is 'd'.
$x_{i,14} \sim x_{i,19}$	`Brand`	Brand of the car: 7 categories. Default is 'Fiat'.
$x_{i,20}$	`Gas`	Car gas: Diesel or Regular. Default is 'Diesel'.
$x_{i,21} \sim x_{i,29}$	`Region`	Policy region in France: 10 categories. Default is 'Aquitaine'.
Response	Name	Description
y_{i1}	`ClaimAmount`	Claim amount of the policyholder.

Table 6: Description of French Auto Insurance Data

4.1. Data Description and Pre-proccecing

We consider the French auto-insurance claims dataset included in the R package `CASdatasets` (Dutang and Charpentier 2019), which can be loaded as follows.

```
> library(CASdatasets)
> data(freMTPLfreq, freMTPLsev)
```

The `freMTPLfreq` dataset contains records of policy ID, policyholder information and number of claims, while `freMTPLsev` contains only policy ID and claim amounts for those who have claims. The `PolicyID` variable serves as a unique identifier which links these two datasets. Some policyholders have multiple claims, resulting in dulplicate records of `PolicyID` in `freMTPLsev`. For demonstration purposes, we aggregate claim amounts by policyholder, and choose only the claim severity as response variable.

```
# Aggregate claim amounts
> sev.aggre = aggregate(freMTPLsev$ClaimAmount,
                        by = list(PolicyID = freMTPLsev$PolicyID),
                        FUN = "sum")
> colnames(sev.aggre)[2] = "ClaimAmount"

# Match two datasets by PolicyID. NA values correspond to no claims.
> df.all = merge(freMTPLfreq, sev.aggre, by = "PolicyID", all = TRUE)
> df.all$ClaimAmount[is.na(df.all$ClaimAmount)] = 0
```

The resulting dataset has 413,169 observations. The claim amount has high zero inflation, as less than 4% of policyholders have filed at least one claim. For positive claim amounts, the distribution is right-skewed, multi-modal and heavy-tailed. The variables in the aggregated dataset are described in Table 6.

Standard R functions are used to convert the above dataframe into matrices covariates and response required by the **LRMoE** package.

```
# Make Y matrix
> sample.size = nrow(df.all)
> Y = matrix( c(rep(0, sample.size),    # = tl
                df.all$ClaimAmount,     # = yl
```

```
                df.all$ClaimAmount,     # = yu
                rep(Inf, sample.size)   # = tu
                ),
              ncol = 4, byrow = FALSE)

# Make X matrix
> X.continuous = cbind(df.all$CarAge, df.all$DriverAge)
> X.power = model.matrix(~df.all$Power, data = df.all)   # Default is 'd'
> X.brand = model.matrix(~df.all$Brand, data = df.all)   # Default is 'Fiat'
> X.gas = model.matrix(~df.all$Gas, data = df.all)       # Default is 'Diesel'
> X.region = model.matrix(~df.all$Region, data = df.all) # Default is 'Aquitaine'
> X = matrix(cbind(rep(1, sample.size), # Intercept
                   X.continuous,
                   X.power[,-1], X.brand[,-1], X.gas[,-1], X.region[,-1]),
              nrow = sample.size, byrow = FALSE)
# Omitted code to name columns of X: colnames(X) = c(...)
```

4.2. Parameter Initialization

Since the fitting procedure of LRMoE involves multivariate optimization, a good initialization of parameters will often lead to faster convergence, compared with a noninformative guess. Gui *et al.* (2018) have proposed an initialization procedure for fitting mixture of Erlang distributions, which involves k-means clustering and clusterized method of moments (CMM). This has been used in Fung *et al.* (2020b) and offers reasonably good starting values of parameters.

Our package contains an initialization function which applies the CMM method to all component distributions, which is used in conjunction with the base function `kmeans`. Some preliminary analysis is needed to determine the number of clusters (components) to use. Since the positive part of all distributions included in our package is unimodal, a heuristic starting point is to examine the empirical histogram of data and count the number of peaks (see Figure 3). As an example, the procedure to initialize a 3-component LRMoE is given as follows.

```
# Number of components
> n.comp = 3

# Perform k-means clustering on normalized data and covariates: df.all.st
# In case of censored data, mid-point of censoring bounds is used
# Code for data normalization is omitted
> km.cluster.3 = kmeans(data.matrix(df.all.st), n.comp)

# Compute parameter initialization using CMM for severity
# Only the labels 'cluster' are needed from the kmeans object
> init.3 = cluster.mm.severity(df.all$ClaimAmount, km.cluster.3$cluster)
```

The `cluster.mm.severity` function will return a list, where each element contains the cluster proportion, zero inflation and parameter initialization for component distributions. The

Table 7: Sample Initialization of Parameters

Component		1	2	3
Proportion		0.21	0.25	0.54
Zero Inflation		0.96	0.97	0.96
Positive Data:				
- Mean		1736.80	2052.66	2487.96
- Coefficient of Variation		2.21	3.00	11.26
- Skewness		10.50	12.54	57.72
- Kurtosis		143.77	193.40	3814.04
- Parameter Initialization				
`gamma`	`shape`	0.20	0.01	0.008
	`scale`	8509.01	18474.62	315719.40
`lnorm`	`meanlog`	6.57	6.48	5.39
	`sdlog`	1.33	1.52	2.20
`invgauss`	`mean`	1736.80	2052.66	3487.96
	`scale`	354.51	228.06	19.61
`weibull`	`shape`[1]	3	3	3
	`scale`	2605.21	3078.99	3731.94
`burr`	`shape1`[1]	2	2	2
	`shape2`[1]	5	5	5
	`scale`	12770.62	15093.07	18293.83

[1] Set to constant for numerical stability. The scale parameter is obtained by matching the first moment.

initialization can be inspected using standard R methods.

The list `init.3` returned by the sample code is summarized in Table 7. The proportion of each cluster and their zero inflation may be used for initializing `alpha.init` and `zero.init`. The summary statistics of positive data are provided to help the user choose a combination of component distributions. Initialization with extremely large or small parameter values should be avoided (e.g. Gamma(0.008, 315714.40) for component 3).

```
# All components have a quite large proportion of zero,
# so component distributions should be zero-inflated.
> comp.dist = matrix(c("ZI-lnorm", "ZI-lnorm", "ZI-lnorm"),
                     nrow = 1, byrow = TRUE)

# Assuming no knowledge of covariate influence on response,
# initialize only the intercept coefficient, where 3 is the reference group
> alpha.init = matrix(0, nrow = 3, ncol = 30)
> alpha.init[,1] = log(c(0.21, 0.25, 0.54)) - log(0.54)

# Initialize zero inflation: all components are zero-inflated.
> zero.init = matrix(0, nrow = 1, ncol = 3)
> zero.init[1,] = c(0.96, 0.97, 0.96)

# Initialize component distribution parameters.
```

```
> params.init = list( list(c(6.57, 1.33), c(6.48, 1.52), c(5.39, 2.20)) )
```

4.3. Parallel Computation

While the LRMoE model provides much flexibility, the user needs to try different combinations of component distributions and/or different initialization of parameters in order to find a model which best fits the data. If the fitting function is called sequentially for each model speficication, the computational time scales with the number of LRMoE model candidates, which is not desirable especially for large datasets. In this subsection, we will demonstrate how to integrate our package with `doParallel` (Corporation and Weston 2019), so that multiple LRMoE models can be simultaneously fitted.

We assume that the user has specified a collection of LRMoE model candidates. Each of them is represented as a list, which contains a string `model.name` as an identifier, model specification and parameter initialization as described in Section 3.

```
# The model.list contains a list of LRMoE model candidates to fit.
# model.list = list(model.1, model.2, ...)

# Each model is structured as a list of inputs
# required by the fitting function.
# model.1 = list(model.name, n.comp, comp.dist,
                 alpha.init, zero.init, params.init,
                 hyper.alpha, hyper.params)
```

The fitting function will be called in a loop function `do_fitting`, where each iteration outputs one LRMoE model. For ease of analysing results, it is recommended to save the fitted model as a `.Rda` file, as well as the intermediate output as a `.txt` file. Additionally, the user may opt to get email updates on the running status if the code is running on a remote server (see e.g. Premraj 2015).

```
do_fitting = function(X, Y, model)
{
  model.name = toString(paste(model$model.name, sep=""))
  rda.name = toString(paste(model.name, ".Rda", sep=""))
  output.name = toString(paste(model.name, ".txt", sep=""))

  # Set a file to save the intermediate update of parameters
  sink(file = output.name)

  tryCatch({model.fit = LRMoE.fit(Y = Y, X = X,
              n.comp = model$n.comp, comp.dist = model$comp.dist,
              alpha.init = model$alpha.init, zero.init = model$zero.init,
              params.init = model$params.init,
              penalty = TRUE,
              hyper.alpha = model$hyper.alpha, hyper.params = model$hyper.params,
              print = TRUE)
            save(model.fit, file = rda.name) # Save fitted model
```

```
          },
          error=function(e){"Error!"; print("Error!")}
         )

  # Save intermediate update of parameters for the current model
  sink()

  # Optional: use mailR to get running status. Code is omitted.
}
```

Finally, the `do_fitting` function is called in parallel. Depending on the user's computing resources, the number of models to fit in parallel (`ncore`) may be different.

```
# Specify how many models to fit in parallel
> ncore = 5
> n.run = length(model.list)

# Make computing clusters: standard procedure
> cl = makePSOCKcluster(ncore)
> registerDoParallel(cl)

# Call fitting functions in parallel
> foreach(b = 1:n.run) %dopar% {do_fitting(X, Y, model.list[[b]])}

# Stop computing clusters: standard procedure
> stopCluster(cl)
```

4.4. Fitting Results and Model Selection

For illustration purposes, we fit only a selected number of LRMoEs to the entire French auto insurance dataset. The fitted loglikelihood, AIC and BIC are summarized in Table 8. In most cases, adding more components will increase the fitted loglikelihood (e.g. consider models `iwl`, `will`, `wllil` and `liwlll`). The models selected by AIC and BIC have 6 and 4 components, respectively. In this particular setting, BIC heavily penalizes models with more components, since the sample size is large, and adding one component roughly increases the number of parameters by 30 (the number of logit regression coefficients).

Apart from AIC and BIC, cross validation (CV) is an alternative model selection criterion which avoids overfitting with too many latent components. For example, Gui *et al.* (2018) consider a 10-fold CV for fitting mixture of Erlangs, where the averaged loglikelihood on the test sets is used as a score function to select the optimal number of components.

To implement CV, our package provides a function to calculate the loglikelihood of a fitted model on a test dataset, which can be incorporated in the `do_fitting` function above with additional input of test sets `X.test` and `Y.test`.

```
# X.test, Y.test: Test sets formatted as required by LRMoE
# model.fit: A fitted model returned by LRMoE.fit function
> LRMoE.loglik(X = X.test, Y = Y.test, model = model.fit)
```

Table 8: Fitting Results of French Auto Insurance Data

$g = 3$	Loglikelihood	AIC	BIC	$g = 4$	Loglikelihood	AIC	BIC
`lll`	-183913	367965	368719	`lill`	-183521	367246	**368361**
`iwl`	-183928	367993	368748	`llll`	-183749	367702	368817
`wll`	-184078	368293	369047	`bill`	-183784	367774	368900
`llb`	-184099	368337	369102	`will`	-183798	367800	368915
`lib`	-184112	368363	369129	`bilw`	-183954	368114	369240
$g = 5$	Loglikelihood	AIC	BIC	$g = 6$	Loglikelihood	AIC	BIC
`bwliw`	-183576	367424	368910	`llblll`	**-183444**	**367226**	369074
`bllil`	-183590	367452	368938	`liwibl`	-183450	367238	369085
`lllil`	-183603	367476	368951	`llwlll`	-183466	367269	369105
`llll`	-183604	367478	368954	`liwlll`	-183471	367279	369115
`wllil`	-183654	367578	369054	`liblll`	-183491	367321	369168

[1] Component distributions are represented by first letter, e.g. `l` for lognormal, `b` for Burr, etc.
[2] Stopping criterion is < 0.05 improvement in loglikelihood, or 500 iterations.
[3] For each g, loglikelihood values are in decreasing order. The overall optimal values are bolded.

4.5. Pricing and Reserving Functions

Our 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) $E[(Y \wedge u)]$ and stop-loss (SL) premium $E[(Y - d)_+]$ 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.

As an illustration, we consider three policyholders in Table 9, where A has no claim history, B has a medium-sized claim and C has a large claim. Below is the sample code to calculate different quantities of interest.

```
# 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

# 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
```

	Claim	ID	Car Age	Driver's Age	Car Power	Brand	Gas	Region
A	0	1	0	46	g	JK	Diesel	A
B	302	33	1	61	g	JK	Regular	IF
C	10870	96	0	51	j	JK	Regular	IF

Table 9: Selected Policyholders in French Auto Insurance Dataset

	Premium Principle	A	B	C
Individual-Level	$E[Y]$	97.49	90.25	84.65
	$E[(Y-1000)_+]$	78.76	69.81	58.29
	$E[Y \wedge 100000]$	63.24	60.86	63.33
Portfolio-Level	VaR(90)	108.65	100.59	94.34
	VaR(95)	121.14	112.15	105.19
	CTE(70)	117.20	108.51	101.77
	CTE(80)	126.58	117.19	109.91
	CTE(90)	149.28	138.20	129.62

Table 10: Pricing calculation for selected policyholders in French Auto Insurance Dataset. Calculation is based on covariates and the fitted model only, i.e. not considering claim history. For portfolio-level premium principles, the allocation is based on the relative size of pure premium, where the weight for policyholder i is $w_i = E[Y_i]/\sum_j E[Y_j]$.

```
[2,] 1203.652
[3,] 1249.037

# Mean excess of claim amount (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
```

Actuarial pricing calculation can be done based on either individual loss distributions, or the aggregated loss distribution of the entire portfolio. Both can be achieved using built-in functions in our package.

On the individual level, the functions above can be called directly to calculate the pure premium $E[Y]$, as well as the premium value in the presence of a policy deductible ($E[(Y-d)_+]$) or policy limit ($E[(Y \wedge u)]$). The first part of Table 10 summarizes the premium calculation for Policyholders A, B and C, based on individual-level premium principles.

On the portfolio level, the `dataset.simulator` function (see also Section 4.6) will simulate one response value (i.e. claim amount) for each individual policyholder, which can be summed up to the aggregated portfolio loss under one possible scenario. Repeated simulation of the entire portfolio will produce the empirical distribution for the aggregated loss. The VaR and CTE of the aggregated loss can be obtained from the simulated sample, which are useful

for setting the insurer's total reserve. In addition, the VaR and CTE can be allocated back to policyholders as a loaded premium, according to some weighting scheme which reflects policyholders' relative riskiness (e.g. relative magnitude of their pure premium). The second part of Table 10 summarizes the premium calculation for Policyholders A, B and C, based on portfolio-level premium principles.

4.6. Model Visualization

After fitting and choosing an appropriate LRMoE model, the user can visualize it with in-package plotting functions, or create more customized plots using generic simulation functions (e.g. `data.similator`) combined with base R plotting utilities or other dedicated packages (e.g. `ggplot2`). In this subsection, we will use the 6-component `llblll` model for demonstration.

Latent Class Probabilities

The logit regression in LRMoE assigns each policyholder into latent risk classes based on covariates. Given a fitted mode and a vector of covariates, the probability of latent classes can be computed and visualized using the built-in `predict.class.prob` and `plot.ind.class.prob` functions.

For policyholders with known claim history, it may be more informative to consider the posterior latent class probabilities by calling corresponding functions for posterior probabilities. Consider again the policyholders in Table 9. The code to calculate and plot posterior probabilities is shown below, and the plots are given in Figure 2.

```
# 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.posterior.prob(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

# Plot latent class probabilities for Policyholder A
# Function returns a ggplot2 object: optional to edit plot title
> plot.ind.posterior.prob(X.1, Y[1,], model.fit$alpha.fit, model.fit$comp.dist,
+       model.fit$zero.fit, model.fit$params.fit) +
+       ggtitle("Policyholder A")
```

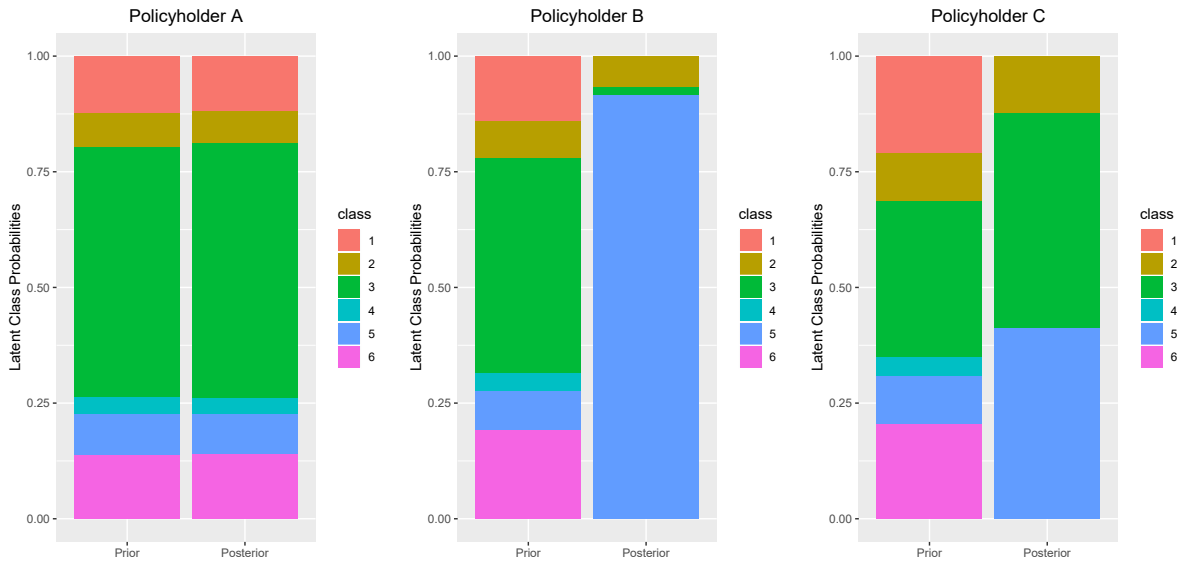

Figure 2: Prior and posterior latent class probabilities for selected policyholders. Component 3 (Burr(1.41, 1, 24073)) corresponds to the tail, while Component 5 (Lognormal(6.23, 1.57)) corresponds to the largest spike in the dataset.

Overall Goodness-of-Fit

The overall goodness-of-fit can be examined by contrasting the empirical histogram against fitted density curve and the Q-Q plot of empirical and fitted quantiles. These can be produced with the `data.simulator` function included in our package, combined with basic R plotting functions. The corresponding plots are shown in Figure 3.

```
# Simulate exact responses, given a set of covariates and a fitted model
> 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]

# 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)

# Use standard R functions for Q-Q plots
> 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)
```

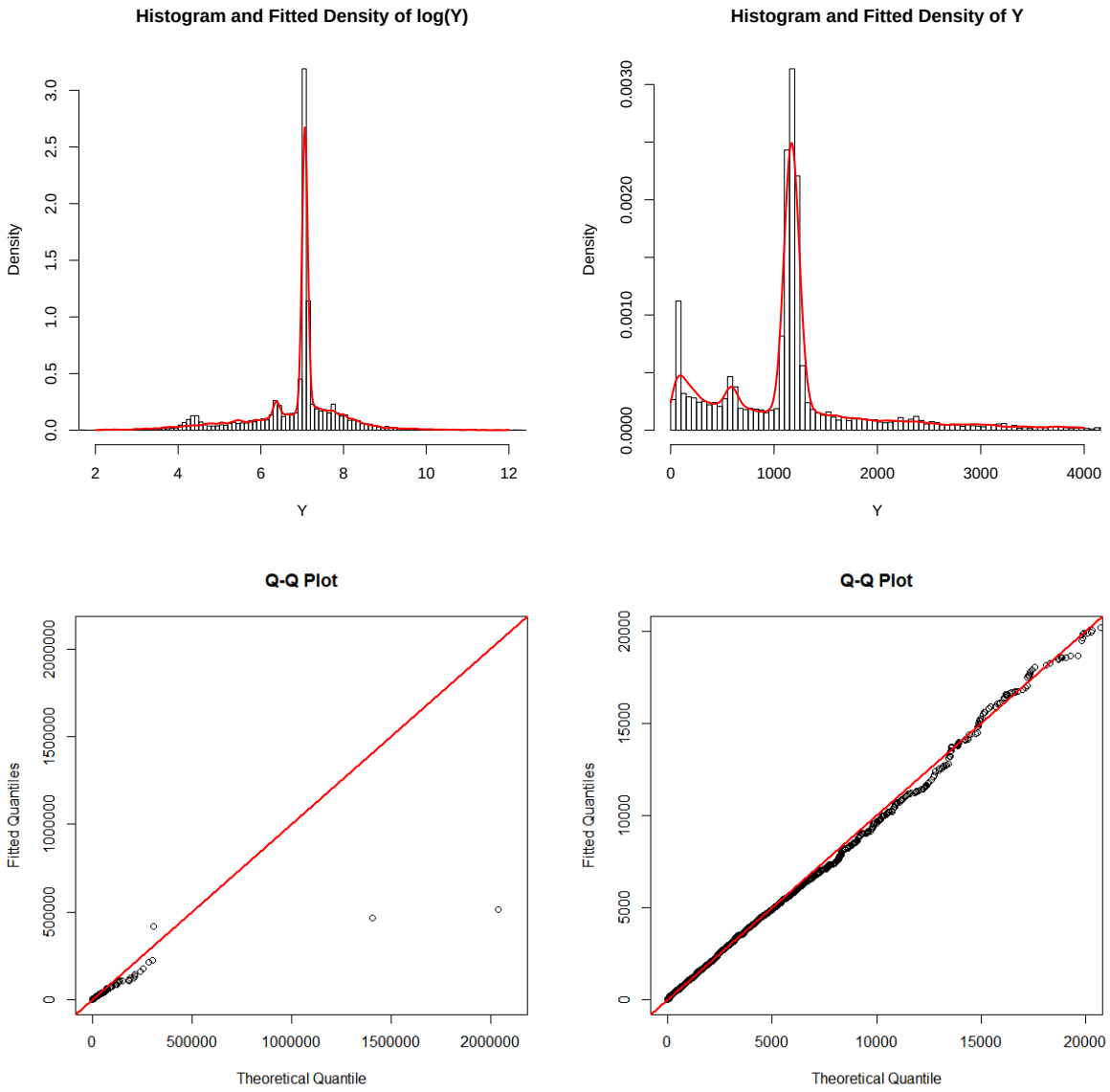

Figure 3: Overall goodness of fit of positive claims in the French Auto Insurance Dataset. Left: all data points. Right: non-extreme data points.

Covariate Influence

The partial dependence plot is commonly used in machine learning to investigate the influence of a particular covariate on the response, assuming independence among covariates (Friedman (2001)). For example, the marginal effect of a covariate on the mean claim amount can be obtained using the following steps:

(1) Fix the covariate at a particular value (say, car brand = 'F') for all policyholders, while other covariates remain unchanged;

(2) Use `predict.mean` function to compute mean response values by policyholder (see Section 4.5);

(3) Compute the grand mean of all values in step (2), which averages out the effect of other covariates and yields the mean response value when the car brand is of type 'F';

(4) Repeat steps (1) – (3) for a range of values of the same covariate of interest, e.g. varying car brand from 'F' to 'VAS'.

The procedure above can be generalized to continuous covariates and other quantities of interest, such as the quantile of the response variable. The covariate influence plot graphically illustrates how the characteristics of a policyholder are associated with a particular measure of the response variable. For example, plotting the mean (or VaR/CTE) of the response variable against car brand reflects how a policyholder's overall riskiness (or tail risk) is related to the car brand.

The calculations above can be done by manually writing loops of `predict.mean`, or by calling the `covinf.discrete` or `covinf.continuous` function in the package which automates this procedure.

```
# Calculate the influence of a chosen discrete covariate
# on some quantity of interest of the response Y.
> BrandInf = covinf.discrete(idx = c(15:20), # column index of the covariate
+                response = c("Mean", "VAR990", "CTE990"),
+                model.fit$alpha.fit, model.fit$comp.dist,
+                model.fit$zero.fit, model.fit$params.fit)
```

The function will return a data frame as shown in Table 11. Standard R data maninpulation with `reshape2` (Wickham 2007) combined with `ggplot2` (Wickham 2016) will produce a plot describing the influence of covariate on the response variable.

Table 11: Sample output of `covinf.discrete` function

Brand	Mean	VaR(99)	CTE(99)
F	86.34	1336.12	5913.98
JK	91.48	1204.76	7999.49
MCB	83.41	1381.02	5684.49
OGF	83.85	1444.18	5422.59
Other	91.44	1352.56	6583.96
RNC	85.10	1329.38	5972.67
VAS	83.67	1421.00	5583.46

```
# Reshaping data
> BrandInf.plot = melt(data = BrandInf, id.vars = c(1), measure.vars = c(2:4))

# Use ggplot2 for plotting
> ggplot(BrandInf.plot, aes(x = Brand, y = value,
+           color = variable, group = variable)) +
+           geom_point(size = 2, show.legend = FALSE) +
+           geom_line(size = 1, show.legend = FALSE) +
+           facet_wrap(.~ variable, scales = "free", ncol = 3) +
+           xlab("") + ylab("") + ggtitle("Covariate Influence: Brand") +
+           theme(plot.title = element_text(hjust = 0.5))
```

Figure 4: Covariate Influence

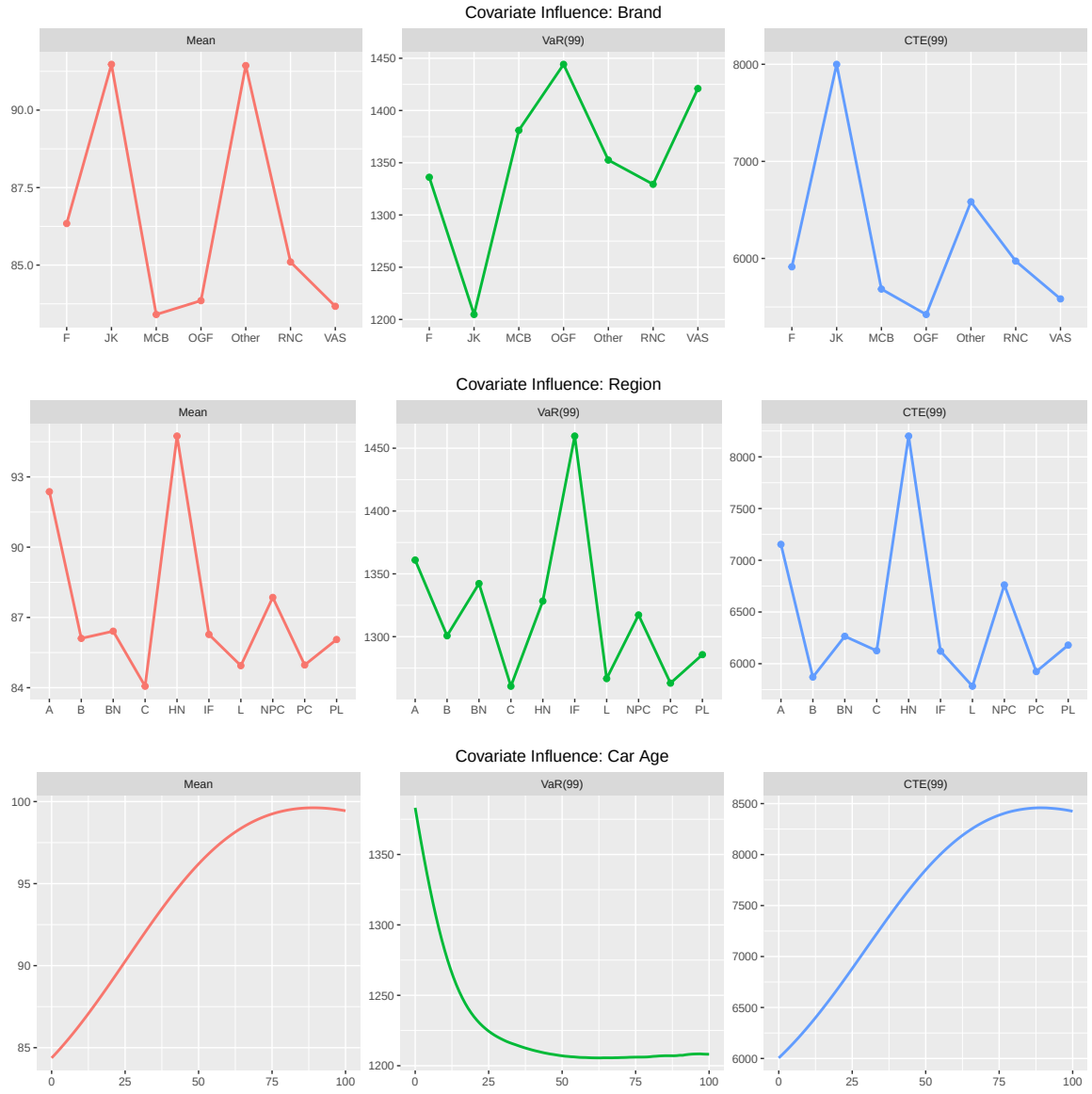

Figure 4 shows the influence of covariates Brand, Region and Car Age, which may be interpreted as follows. For car brand, JK (Japanese except Nissan or Korean) is generally riskier. In terms of tail risk, it is interesting to observe that JK has the lowest VaR but the highest CTE compared with others. Also, compared with other regions, policies issued in regions A (Aquitaine) and HN (Haute-Normandie) may be considered riskier. In addition, older cars are generally associated with higher risks.

5. Summary and Outlook

This paper presented a new R package **LRMoE** for actuarial loss modelling. In this first version, the package can cover the most basic need to fit an LRMoE model, as well as help

the user visualize the fitted model and calculate insurance premium. There are several future developments in our plan, including:

- Functions to address uncertainties in parameter estimation: Currently the fitting function only returns point estimates of model parameters. It may be more insightful to also provide a measure of uncertainty in these estimates.
- Model selection tools: As of the current version, model selection is done by AIC/BIC/CV, all of which require the user to choose and run a selection of model. Some automated model selection procedures may be potentially incorporated in the package (e.g. SCAD type penalty in Fan and Li 2001 and Yin and Lin 2016).
- Feature selection tools: Datasets usually contain a large number of covariates, but not all are important for predicting the response. While plots of covariate influence may offer some intuition of their relative importance, it may be more insightful to quantify their influence and provide a function to automatically choose the most influential covariates.
- Optimization of internal code structure for better computation speed: The runtime on a single core for fitting a 4-component LRMoE model in Section 4 is around 5 hours. This may be improved by internally changing some calculation methods, rewriting time-consuming procedures in faster languages such as C++, or potentially using parallel computing in each single fitting iteration.
- User interface: In Section 4, the data pre-processing step requires the user to manually convert the data into a specific format required by the package. This may be better completed by adding generic pre-processing functions into the package, or enabling formula interpretation (such as `LRMoE.fit(formula = y ~ x)`).

References

Blostein M, Miljkovic T (2019). "On modeling left-truncated loss data using mixtures of distributions." *Insurance: Mathematics and Economics*, **85**, 35 – 46. ISSN 0167-6687.

Corporation M, Weston S (2019). *doParallel: Foreach Parallel Adaptor for the 'parallel' Package*. R package version 1.0.15, URL https://CRAN.R-project.org/package=doParallel.

Dempster AP, Laird NM, Rubin DB (1977). "Maximum Likelihood from Incomplete Data Via the EM Algorithm." *Journal of the Royal Statistical Society: Series B (Methodological)*, **39**(1), 1–22.

Dutang C, Charpentier A (2019). *CASdatasets: Insurance datasets*. R package version 1.0-10.

Fan J, Li R (2001). "Variable selection via nonconcave penalized likelihood and its oracle properties." *Journal of the American statistical Association*, **96**(456), 1348–1360.

Friedman JH (2001). "Greedy function approximation: a gradient boosting machine." *Annals of statistics*, pp. 1189–1232.

Fung TC, Badescu AL, Lin XS (2019a). "A Class of Mixture of Experts Models for General Insurance: Application to Correlated Claim Frequencies." *ASTIN Bulletin*, **49**(3), 647688.

Fung TC, Badescu AL, Lin XS (2019b). "A Class of Mixture of Experts Models for General Insurance: Theoretical Developments." *Insurance: Mathematics and Economics*, **89**, 111–127.

Fung TC, Badescu AL, Lin XS (2020a). "Fitting censored and truncated regression data using the Mixture of Experts models." Working paper.

Fung TC, Badescu AL, Lin XS (2020b). "A New Class of Severity Regression Models with an Application to IBNR Prediction." *North American Actuarial Journal*. Forthcoming.

Grün B, Leisch F (2008). "FlexMix Version 2: Finite Mixtures with Concomitant Variables and Varying and Constant Parameters." *Journal of Statistical Software*, **28**(4), 1–35.

Gui W, Huang R, Lin XS (2018). "Fitting the Erlang mixture model to data via a GEM-CMM algorithm." *Journal of Computational and Applied Mathematics*, **343**, 189–205.

Jiang W, Tanner MA (1999). "On the identifiability of mixtures-of-experts." *Neural Networks*, **12**(9), 1253–1258.

Jordan MI, Jacobs RA (1994). "Hierarchical mixtures of experts and the EM algorithm." *Neural computation*, **6**(2), 181–214.

Lee D, Li WK, Wong TST (2012). "Modeling insurance claims via a mixture exponential model combined with peaks-over-threshold approach." *Insurance: Mathematics and Economics*, **51**(3), 538–550.

Leisch F (2004). "FlexMix: A General Framework for Finite Mixture Models and Latent Class Regression in R." *Journal of Statistical Software*, **11**(8), 1–18.

McLachlan G, Peel D (2004). *Finite Mixture Models*. John Wiley & Sons.

Meng XL, Rubin DB (1993). "Maximum likelihood estimation via the ECM algorithm: A general framework." *Biometrika*, **80**(2), 267–278. ISSN 0006-3444.

Miljkovic T, Grün B (2016). "Modeling loss data using mixtures of distributions." *Insurance: Mathematics and Economics*, **70**, 387 – 396. ISSN 0167-6687.

Premraj R (2015). *mailR: A Utility to Send Emails from R*. R package version 0.4.1, URL https://CRAN.R-project.org/package=mailR.

Scollnik DP, Sun C (2012). "Modeling with weibull-pareto models." *North American Actuarial Journal*, **16**(2), 260–272.

Wickham H (2007). "Reshaping Data with the reshape Package." *Journal of Statistical Software*, **21**(12), 1–20. URL http://www.jstatsoft.org/v21/i12/.

Wickham H (2016). *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York. ISBN 978-3-319-24277-4. URL https://ggplot2.tidyverse.org.

Yin C, Lin XS (2016). "Efficient estimation of Erlang mixtures using iSCAD penalty with insurance application." *ASTIN Bulletin: The Journal of the IAA*, **46**(3), 779–799.