Journal of Statistical Software

February 2018, Volume 83, Issue 7. *doi: 10.18637/jss.v083.i07*

teigen: An R Package for Model-Based Clustering and Classification via the Multivariate t Distribution

Jeffrey L. Andrews
University of British Columbia

Jaymeson R. Wickins
MacEwan University

Nicholas M. Boers
MacEwan University

Paul D. McNicholas
McMaster University

Abstract

The **teigen** R package is introduced and utilized for model-based clustering and classification. The tEIGEN family of mixtures of multivariate t distributions is formed via an eigen-decomposition of the component covariance matrices and subsequent component-wise constraints. The **teigen** package implements all previously published tEIGEN family members as well as eight additional models: four multivariate and four univariate. The resulting family of 32 mixture models is implemented in both serial and parallel, with useful dedicated functions. Methodology and examples that illustrate **teigen**'s functionality are presented.

Keywords: mixture models, classification, model-based clustering, cluster analysis, multivariate t distributions, **teigen**, R.

1. Introduction

The usage of mixture models for cluster analysis is commonly referred to as model-based clustering. A random vector $\mathbf{X}$ arises from a parametric finite mixture model if the density of origin can be written as $f(\mathbf{x} \mid \boldsymbol{\vartheta}) = \sum_{g=1}^{G} \pi_g p_g(\mathbf{x} \mid \boldsymbol{\theta}_g)$, where $\boldsymbol{\vartheta}$ is the parameter space, G is the number of components, π_g are the mixing weights such that $\sum_{g=1}^{G} \pi_g = 1$ and all $\pi_g > 0$, and $p_g(\mathbf{x} \mid \boldsymbol{\theta}_g)$ are parametric densities with parameters $\boldsymbol{\theta}_g$.

The multivariate Gaussian distribution has received the bulk of researchers' attention over the past couple of decades (e.g., Banfield and Raftery 1993; Celeux and Govaert 1995; Fraley and

Raftery 2002; McNicholas and Murphy 2008). More recently, non-Gaussian mixture models have attracted the attention of researchers (e.g., McLachlan and Peel 1998; Lin 2010; Karlis and Santourian 2009; Andrews and McNicholas 2011b,a; Vrbik and McNicholas 2012, 2014; Franczak, Browne, and McNicholas 2014; Browne and McNicholas 2015). The most common method of fitting these models is by using the expectation-maximization algorithm or a closely related variant.

The number of parameters requiring estimation in parametric mixture models can be computationally crippling and in many cases increases quadratically with the dimensionality of the data. The bulk of these parameters lie in the covariance structures of the component densities. One way of reducing this hindrance is the development of families of mixture models that arise from constraints on the covariance structure – indeed, many of the previously noted references are focused on this task. The flagship mixture model family is the set of Gaussian parsimonious clustering models (GPCM), which is derived from the multivariate Gaussian distribution with eigen-decomposed covariance structure (Banfield and Raftery 1993; Celeux and Govaert 1995). All fourteen members of the GPCM family are available in the **Rmixmod** package (Auder, Lebret, Iovleff, and Langrognet 2016) and the **mixture** package (Browne and McNicholas 2016) and have also implemented in a recent update of the popular **mclust** package (Fraley and Raftery 1998; Fraley, Raftery, Murphy, and Scrucca 2012; Fraley, Raftery, Scrucca, Murphy, and Fop 2016). Furthermore, skew-normal, restricted skew-t, unrestricted skew-t, and generalized hyperbolic mixture distributions can be fit via the **EMMIXskew** (Wang, Ng, and McLachlan 2017), **EMMIXuskew** (Lee and McLachlan 2013), and **MixGHD** (Tortora, Browne, Franczak, and McNicholas 2015) packages, respectively.

In the section that follows, we introduce eight new models, a novel closed-form estimation for the degrees of freedom, and review the multivariate t distributional equivalent of the GPCM family – the tEIGEN family (Andrews and McNicholas 2012; Andrews, Wickins, Boers, and McNicholas 2018) – for both clustering and classification. Section 3 includes specifics on the R package introduced, while Section 4 gives examples for application of the package. We conclude in Section 5 with a summary.

2. The tEIGEN family

Andrews and McNicholas (2012) introduce the 24-member tEIGEN mixture model family for model-based clustering and classification. The tEIGEN family is derived from mixtures of multivariate t distributions, whose density is

$$f(\mathbf{x} \mid \boldsymbol{\vartheta}) = \sum_{g=1}^{G} \pi_g f_t(\mathbf{x} \mid \boldsymbol{\mu}_g, \boldsymbol{\Sigma}_g, \nu_g),$$

where $f_t(\mathbf{x} \mid \boldsymbol{\mu}_g, \boldsymbol{\Sigma}_g, \nu_g)$ is the multivariate t density

$$f_t(\mathbf{x} \mid \boldsymbol{\mu}_g, \boldsymbol{\Sigma}_g, \nu_g) = \frac{\Gamma\left(\frac{\nu_g+p}{2}\right)|\boldsymbol{\Sigma}_g|^{-\frac{1}{2}}}{(\pi\nu_g)^{\frac{p}{2}}\Gamma\left(\frac{\nu_g}{2}\right)\left[1+\frac{\delta(\mathbf{x},\boldsymbol{\mu}_g|\boldsymbol{\Sigma}_g)}{\nu_g}\right]^{\frac{\nu_g+p}{2}}},$$

with mean vector $\boldsymbol{\mu}_g$, scale matrix $\boldsymbol{\Sigma}_g$, and degrees of freedom ν_g.

Following the work of Banfield and Raftery (1993) and Celeux and Govaert (1995), an eigen-decomposition is imposed on the scale matrix $\boldsymbol{\Sigma}_g = \lambda_g \mathbf{D}_g \mathbf{A}_g \mathbf{D}_g^\top$, where $\mathbf{D}_g$ is the matrix of

eigenvectors, $\mathbf{A}_g$ is the diagonal matrix of eigenvalues with $|\mathbf{A}_g| = 1$, and λ_g are the associated constants of proportionality. These individual scalars/matrices can then be constrained to be equal across mixture components, or in some cases constrained to be the identity matrix: $\boldsymbol{\Sigma}_g = \lambda_g \mathbf{I}\mathbf{A}\mathbf{I}^\top = \lambda_g \mathbf{A}$, for instance. It is important to note that under certain constraints on this covariance matrix decomposition, the model being fit loses the property of scale invariance. As such, the **teigen** package will scale variables to have mean 0 and variance 1 by default (see Section 3 for controlling this option). In addition, constraints are also imposed on the degrees of freedom ν_g, following Andrews and McNicholas (2011a). Thus, taking all possible combinations of these constraints into consideration would result in a 28-model family (see Table 1), 24 of which were originally developed for the *t*EIGEN family (Andrews and McNicholas 2012). Those 24 models correspond to 12 of the 14 decompositions of $\boldsymbol{\Sigma}_g$ used in the GPCM family along with the constraint for ν_g. The four remaining models, and methodology required for their estimation, are discussed in Section 2.1.

All of the previously considered *t*EIGEN models are applicable only to multivariate data. In order for the package introduced herein to be applicable to univariate data, we introduce four more models into the *t*EIGEN family. The univariate mixture takes the form

$$f(x \mid \boldsymbol{\vartheta}) = \sum_{g=1}^{G} \pi_g f_t(x \mid \mu_g, \sigma_g^2, \nu_g),$$

where μ_g is the component mean and σ_g^2 is the scale parameter. We once again allow constraints on the scale and degrees of freedom, by permitting them to be equal across groups. The four resulting models are summarized in Table 2. Hereafter, we refer to the combination of all 32 univariate and multivariate models as the '*t*EIGEN family'.

2.1. Parameter estimation

As in the majority of mixture modelling implementations, we use a variant of the expectation-maximization (EM) algorithm (Dempster, Laird, and Rubin 1977) to perform parameter estimation. The EM algorithm consists of two steps (expectation and maximization) performed iteratively until convergence. On the E-step, we compute the expected value of the complete-data log-likelihood, and on the M-step, the parameters are maximized according to the complete-data log likelihood. The expectation-conditional maximization (ECM) algorithm (Meng and Rubin 1993) adjusts the M-step to allow several, more efficient, conditional maximization (or CM) steps.

To implement an EM algorithm, we need to be able to compute the likelihood for our mixture models. First, we define a random variable Z_{ig} such that $z_{ig} = 1$ if observation i belongs to group g and otherwise $z_{ig} = 0$. In a model-based classification (or semi-supervised) scenario, we observe the first k observations with known group membership, and the remaining observations, giving n total observations, with unknown group membership. The likelihood can thus be written

$$\mathcal{L}(\boldsymbol{\vartheta}) = \prod_{i=1}^{k} \prod_{g=1}^{G} \left[\pi_g f_t(\mathbf{x}_i \mid \boldsymbol{\mu}_g, \boldsymbol{\Sigma}_g, \nu_g)\right]^{z_{ig}} \times \prod_{j=k+1}^{n} \sum_{h=1}^{H} \pi_h f_t(\mathbf{x}_j \mid \boldsymbol{\mu}_h, \boldsymbol{\Sigma}_h, \nu_h). \tag{1}$$

In a model-based clustering scenario, none of the group memberships are known, and therefore the likelihood is simply the right hand side of Equation 1 with $k = 0$ and $H = G$. Note that

Model	λ_g	$\mathbf{D}_g$	$\mathbf{A}_g$	ν_g	Free covariance and df parameters
CIIC	C	I	I	C	$1+1$
CIIU	C	I	I	U	$1+G$
UIIC	U	I	I	C	$G+1$
UIIU	U	I	I	U	$G+G$
CICC	C	I	C	C	$p+1$
CICU	C	I	C	U	$p+G$
UICC	U	I	C	C	$(p-1)+G+1$
UICU	U	I	C	U	$(p-1)+G+G$
CIUC	C	I	U	C	$Gp-(G-1)+1$
CIUU	C	I	U	U	$Gp-(G-1)+G$
UIUC	U	I	U	C	$Gp+1$
UIUU	U	I	U	U	$Gp+G$
CCCC	C	C	C	C	$[p(p+1)/2]+1$
CCCU	C	C	C	U	$[p(p+1)/2]+G$
UCCC	U	C	C	C	$[p(p+1)/2]+(G-1)+1$
UCCU	U	C	C	U	$[p(p+1)/2]+(G-1)+G$
CUCC	C	U	C	C	$G[p(p+1)/2]-(G-1)(p)+1$
CUCU	C	U	C	U	$G[p(p+1)/2]-(G-1)(p)+G$
UUCC	U	U	C	C	$G[p(p+1)/2]-(G-1)(p-1)+1$
UUCU	U	U	C	U	$G[p(p+1)/2]-(G-1)(p-1)+G$
CCUC*	C	C	U	C	$[p(p+1)/2]+(G-1)(p-1)+1$
CCUU*	C	C	U	U	$[p(p+1)/2]+(G-1)(p-1)+G$
CUUC	C	U	U	C	$G[p(p+1)/2]-(G-1)+1$
CUUU	C	U	U	U	$G[p(p+1)/2]-(G-1)+G$
UCUC*	U	C	U	C	$G[p(p+1)/2]+(G-1)(p)+1$
UCUU*	U	C	U	U	$G[p(p+1)/2]+(G-1)(p)+G$
UUUC	U	U	U	C	$G[p(p+1)/2]+1$
UUUU	U	U	U	U	$G[p(p+1)/2]+G$

Table 1: *t*EIGEN model names and the number of covariance and degrees of freedom parameters requiring estimation. 'C' denotes constrained, 'U' denotes unconstrained, and 'I' denotes the identity matrix. '*' denotes the models being introduced in this manuscript.

there are alternative formulations which include information that falls somewhere between clustering and classification – for example, semi-supervised clustering (Melnykov, Melnykov, and Michael 2015) and fractionally supervised clustering (Vrbik and McNicholas 2015) – but these paradigms are not addressed with the **teigen** software.

The CCUC, CCUU, UCUC, and UCUU models were not previously introduced due to issues with their estimation procedure. Following recent work by Browne and McNicholas (2014) that is implemented in the **mixture** package, an iterative majorize-minimize (MM) algorithm is implemented in the estimation of the common eigenvectors. As an illustration, we provide the specifics for the CCUC model where $\boldsymbol{\Sigma}_g = \lambda \mathbf{D}\mathbf{A}_g\mathbf{D}^\top$.

E-step (CCUC)

The E-step involves computing the conditional expected value of the component indicator

Model	σ_g^2	ν_g	Free variance and df parameters
univCC	C	C	$1+1$
univCU	C	U	$1+G$
univUC	U	C	$G+1$
univUU	U	U	$G+G$

Table 2: Univariate model names and the number of variance and degrees of freedom parameters requiring estimation. 'C' denotes constrained, 'U' denotes unconstrained.

variables Z_{ig} and the characteristic weights W_{ig}:

$$\mathbb{E}[Z_{ig} \mid \mathbf{x}_i] = \frac{\pi_g f_t(\mathbf{x} \mid \boldsymbol{\mu}_g, \lambda \mathbf{D}\mathbf{A}_g\mathbf{D}^\top, \nu_g)}{\sum_{h=1}^G \pi_h f_t(\mathbf{x} \mid \boldsymbol{\mu}_h, \lambda \mathbf{D}\mathbf{A}_h\mathbf{D}^\top, \nu_h)} =: \hat{z}_{ig},$$

$$\mathbb{E}[W_i \mid \mathbf{x}_i, Z_{ig} = 1] = \frac{\nu_g + p}{\nu_g + \delta(\mathbf{x}_i, \boldsymbol{\mu}_g \mid \lambda \mathbf{D}\mathbf{A}_g\mathbf{D}^\top)} =: \hat{w}_{ig}.$$

The $\hat{z}_{ig}$ represent the probability that observation i belongs to group g given the current component parameters. The $\hat{w}_{ig}$ can be thought of as a weight for how much influence observation i has on the estimation of $\boldsymbol{\mu}_g$ and $\boldsymbol{\Sigma}_g$. This interpretation may become more clear in the section that follows while viewing the updates for $\hat{\boldsymbol{\mu}}_g$ and the sample covariance matrix $\mathbf{S}_g$.

CM-steps (CCUC)

The CM-steps involve conditionally maximizing the parameters with respect to the complete-data log-likelihood. In the first of two CM-steps, the mixing proportions, component means, and degrees of freedom are updated:

$$\hat{\pi}_g = \frac{n_g}{n} \quad \text{and} \quad \hat{\boldsymbol{\mu}}_g = \frac{\sum_{i=1}^n \hat{z}_{ig}\hat{w}_{ig}\mathbf{x}_i}{\sum_{i=1}^n \hat{z}_{ig}\hat{w}_{ig}},$$

where $n_g = \sum_{i=1}^n \hat{z}_{ig}$. We discuss a novel alternative estimation method for the degrees of freedom in Section 2.2. In the second CM-step, the decomposed elements of the covariance matrix are updated according to the algorithm that follows, where $\mathbf{S}_g = (1/n_g)\sum_{i=1}^n \hat{z}_{ig}\hat{w}_{ig}(\mathbf{x}_i - \hat{\boldsymbol{\mu}}_g)(\mathbf{x}_i - \hat{\boldsymbol{\mu}}_g)^\top$.

1. Iteration $t = 1$.
2. Update $\hat{\lambda}$:
$$\hat{\lambda} = \frac{\sum_{g=1}^G \operatorname{tr}\{n_g \mathbf{S}_g \hat{\mathbf{D}}\hat{\mathbf{A}}_g^{-1}\hat{\mathbf{D}}^\top\}}{np}.$$
3. If t is odd, update dummy matrix $\mathbf{U}$ according to (a), else update according to (b). Here s_g is the largest eigenvalue of $n_g\mathbf{S}_g$ and a_g is the largest eigenvalue of $(\hat{\lambda}\hat{\mathbf{A}}_g)^{-1}$.
   (a) $\mathbf{U} = \sum_{g=1}^G n_g(\hat{\lambda}\hat{\mathbf{A}}_g)^{-1}(\hat{\mathbf{D}}^{\text{old}})^\top\mathbf{S}_g - s_g(\hat{\lambda}\hat{\mathbf{A}}_g)^{-1}(\hat{\mathbf{D}}^{\text{old}})^\top$.
   (b) $\mathbf{U} = \sum_{g=1}^G n_g\mathbf{S}_g\mathbf{D}(\hat{\lambda}\hat{\mathbf{A}}_g)^{-1} - a_g n_g \mathbf{S}_g\mathbf{D}$.
4. Perform singular value decomposition on $\mathbf{U} = \mathbf{P}\mathbf{B}\mathbf{R}^\top$.

5. Update $\hat{\mathbf{D}} = \mathbf{R}\mathbf{P}^\top$.
6. Update
$$\hat{\mathbf{A}}_g = \frac{\text{diag}(n_g \hat{\mathbf{D}}^\top \mathbf{S}_g \hat{\mathbf{D}})}{|n_g \hat{\mathbf{D}}^\top \mathbf{S}_g \hat{\mathbf{D}}|^{1/p}}.$$
7. Calculate $\mathbf{F}_t = \frac{1}{\hat{\lambda}} \sum_{g=1}^{G} \text{tr}\{n_g \hat{\mathbf{D}} \hat{\mathbf{A}}_g^{-1} \hat{\mathbf{D}}^\top \mathbf{S}_g\} + np \log(\hat{\lambda})$.
8. If $t > 1$, check if $\mathbf{F}_t - \mathbf{F}_{t-1} > \epsilon_1$. If TRUE, $t = t + 1$ and return to step 2, else end.

Note that ϵ_1 is user defined with a default value of 0.001. See the discussion of `eps` in Section 3 for setting this value.

2.2. Degrees of freedom

In recent work (Andrews, McNicholas, and Subedi 2011; Andrews and McNicholas 2011b,a, 2012), the degrees of freedom are updated using numeric estimation of the following non-closed-form equations: for unconstrained degrees of freedom

$$1 - \varphi\left(\frac{\hat{\nu}_g}{2}\right) + \log\left(\frac{\hat{\nu}_g}{2}\right) + \varphi\left(\frac{\hat{\nu}_g^{\text{old}} + p}{2}\right) - \log\left(\frac{\hat{\nu}_g^{\text{old}} + p}{2}\right) + \frac{1}{n_g}\sum_{i=1}^{n} \hat{z}_{ig}(\log \hat{w}_{ig} - \hat{w}_{ig}) = 0, \quad (2)$$

is solved for $\hat{\nu}_g$, while constraining across groups leads to the following equation

$$1 - \varphi\left(\frac{\hat{\nu}}{2}\right) + \log\left(\frac{\hat{\nu}}{2}\right) + \varphi\left(\frac{\hat{\nu}^{\text{old}} + p}{2}\right) - \log\left(\frac{\hat{\nu}^{\text{old}} + p}{2}\right) + \frac{1}{n}\sum_{g=1}^{G}\sum_{i=1}^{n} \hat{z}_{ig}(\log \hat{w}_{ig} - \hat{w}_{ig}) = 0,$$

that is solved for $\hat{\nu}$. In terms of the numeric solution, the `uniroot` function from R is used which is based on the `zeroin` subroutine by Brent (1973).

Herein, we introduce a novel closed-form approximation for constrained degrees of freedom. With $k = -1 - \frac{1}{n}\sum_{g=1}^{G}\sum_{i=1}^{n} \hat{z}_{ig}(\log \hat{w}_{ig} - \hat{w}_{ig}) - \varphi\left(\frac{\hat{\nu}^{\text{old}}+p}{2}\right) + \log\left(\frac{\hat{\nu}^{\text{old}}+p}{2}\right)$, we can then find an approximation with

$$\hat{\nu} \approx \frac{-\exp(k) + 2\exp(k)\left(\exp\left(\varphi\left(\frac{\hat{\nu}^{\text{old}}}{2}\right)\right) - \left(\frac{\hat{\nu}^{\text{old}}}{2} - \frac{1}{2}\right)\right)}{1 - \exp(k)}. \quad (3)$$

If we alternatively define

$$k = -1 - \frac{1}{n_g}\sum_{i=1}^{n} \hat{z}_{ig}(\log \hat{w}_{ig} - \hat{w}_{ig}) - \varphi\left(\frac{\hat{\nu}^{\text{old}} + p}{2}\right) + \log\left(\frac{\hat{\nu}^{\text{old}} + p}{2}\right)$$

then the approximation holds for $\hat{\nu}_g$ (unconstrained degrees of freedom) instead. Further details and justification on this approximation can be found in Appendix A.

2.3. Initialization

EM algorithms for both clustering and classification require either the initialization of the unknown z_{ig} (which uses maximum likelihood estimation to initialize the parameters) or initialization of the model parameters (which uses expected value computations to initialize

the z_{ig}). The **teigen** package makes use of the former, with several built-in options for initialization: `"kmeans"`, `"hard"` random, `"soft"` random, `"emem"` (which is described in the paragraph that follows), and `"uniform"`. The `"uniform"` option, as described by Andrews *et al.* (2011), sets the initial $\hat{z}_{ig} = \frac{1}{G}$ and is only available for classification scenarios because otherwise the algorithm could not progress. The difference between hard and soft random is the nature of the randomly initialized z_{ig}: hard referring to 0's and 1's, while soft takes on random values between 0 and 1, inclusive. As a further alternative, the user can give specific cluster memberships as an initialization, as described in Section 3. The `"emem"` option is based on the "emEM" approach introduced by Biernacki, Celeux, and Govaert (2003). If this initialization is used, then one of the **teigen** models (chosen by the user) is run for a set number of iterations (chosen by the user) for a set number of starts (chosen by the user) based on one of the other initialization methods (`"soft"`, `"hard"`, or `"kmeans"`). Then, the resulting fit that maximizes the log-likelihood is used to initialize the main algorithm.

We note here that the EM algorithm, in the context of multivariate mixture models and particularly as the number of groups increases, is quite prone to converging on local maxima. This, in turn, means that `teigen` can be quite sensitive to starting values. For this reason, the default `"k-means"` initialization will use 50 random starting points, and the `"soft"` or `"hard"` (completely random) initialization methods are not recommended unless used under the 'emem' approach. Users familiar with the previously noted **mclust** package should keep in mind that the `Mclust()` command uses model-based hierarchical clustering to initialize the EM algorithm – thus leading to deterministic results. This experience can be replicated (as in Section 4.4) by using a deterministic clustering method as a custom initialization.

Note that the degrees of freedom must be initialized, and this value is also user-specified; the default value is 50.

2.4. Estimated time remaining

The user has the option of the function returning, on a continual basis, the time the algorithm has run thus far, estimated time remaining, and percent complete on a single line. An underlying procedure for this updates the time estimates after each model has run – 'each model' here refers to each $G \times 28$ model individually (28 referring to the number of multivariate models in the family). The computational overhead for giving the time estimates is minimal: in the range of a second or two under default ($9 \times 28 = 252$ models) settings. Because it is not updated during each EM iteration, longer model fittings will not lead to longer overhead for the time estimates. Setting `verbose = FALSE` will silence the output, should the user so desire.

2.5. Convergence

Convergence in our algorithm is determined by Aitken's acceleration (Aitken 1926), which at iteration t is given by

$$a^{(t)} = \frac{l^{(t+1)} - l^{(t)}}{l^{(t)} - l^{(t-1)}},$$

where $l^{(t-1)}$ refers to the log-likelihood at iteration $t-1$, and so on. Böhning, Dietz, Schaub, Schlattmann, and Lindsay (1994) propose the usage of $a^{(t)}$ to compute an asymptotic estimate

of the log-likelihood at iteration $t+1$ by

$$l_\infty^{(t+1)} = l^{(t)} + \frac{1}{1-a^{(t)}}(l^{(t+1)} - l^{(t)}).$$

We use the stopping criterion

$$l_\infty^{(t+1)} - l^{(t+1)} < \epsilon_2,$$

from Lindsay (1995) for user-specified ϵ_2, with $\epsilon_2 = 0.1$ as the default. See the discussion of argument `eps` in Section 3 for setting this value.

2.6. Model selection

Selecting the "best" model is a challenge within model-based clustering or classification applications, but the common practice (Fraley and Raftery 2002; McNicholas and Murphy 2010; Andrews and McNicholas 2011b; McNicholas 2016) among researchers is the usage of the Bayesian information criterion (BIC, Schwarz 1978), which is calculated as

$$\text{BIC} = 2l(\mathbf{x}, \hat{\boldsymbol{\vartheta}}) - r \log n,$$

where $l(\mathbf{x}, \hat{\boldsymbol{\vartheta}})$ is the maximized log-likelihood, $\hat{\boldsymbol{\vartheta}}$ is the MLE of $\boldsymbol{\vartheta}$, r is the number of free parameters in the model, and n is the total number of observations.

Another model selection technique permitted with the **teigen** package is the integrated completed likelihood (ICL, Biernacki, Celeux, and Govaert 2000). The ICL makes use of the concept of uncertainty rising from the probabilistic nature of model-based clustering. Generally, and also in the case of the **teigen** package, the main result of a clustering/classification algorithm is a vector of group memberships. They are typically hardened to give maximum *a posteriori* (MAP) classifications via

$$\text{MAP}\{\hat{z}_{ig}\} = \begin{cases} 1 & \text{if } \max_g\{z_{ig}\} \text{ occurs in component } g, \\ 0 & \text{otherwise.} \end{cases}$$

The ICL is then calculated via

$$\text{ICL} \approx \text{BIC} + 2\sum_{i=1}^{n}\sum_{g=1}^{G} \text{MAP}\{\hat{z}_{ig}\} \log \hat{z}_{ig},$$

which is essentially the BIC penalized by the amount of classification uncertainty contained in the model.

2.7. Plots

Three graphics are available when plotting an object of class `teigen`. The first is a bivariate marginal contour plot, where the user can specify the desired variates as well as the resolution of the contours. The second plot is an uncertainty plot, where large dots signify large uncertainty in the classification. Because the $\hat{z}_{ig}$ provide the probability that observation i belongs to group g, we often interpret $\max_g\{\hat{z}_{ig}\}$ as the 'certainty' of our classification of observation i. Conversely, we can interpret $1 - \max_g\{\hat{z}_{ig}\}$ as the 'uncertainty' of our classification of observation i. The uncertainty plot is generated by using this number to determine the point size through the argument `cex`. If the user passes `cex` to the plot, it is used for the

size of the smallest point on the graph. Furthermore, the `uncmult` argument can magnify the size differences for better readability if the user desires.

The two plots mentioned above are only available if the **teigen** object was generated with multivariate data. The options `xmarg` and `ymarg` specify which variables from the data set to plot. By default the plot type is a character vector containing both `"contour"` and `"uncertainty"`, but the user may choose to specify just one of these types to plot just one graph. If both types are provided, an interactive menu will be displayed so the user may switch back and forth between both graphs.

The third plot, the univariate density plot, is the default plot for univariate data and an optional plot for multivariate data. For multivariate data, if the user specifies `ymarg = NULL`, the function will plot a marginal univariate density using `xmarg` as the single variable. This plot includes the kernel density estimate from `density()`, the mixture distribution, and the color-coded component densities.

3. Code specifics

The **teigen** package contains the function `teigen()` which allows the user to perform model-based clustering or classification in serial or parallel with some flexibility on the specifics. The function outputs an object of class `teigen`, for which dedicated print, plot, summary, and predict methods are also included. Most code was written and developed in R 3.3.2 (R Core Team 2017), with a couple of functions outsourced to C. We also note that running `teigen` in parallel depends on the **parallel** package (which is now part of the R core software).

3.1. `teigen()`

The `teigen()` function has the following usage:

```
teigen(x, Gs = 1:9, models = "all", init = "kmeans", scale = TRUE,
  dfstart = 50, known = NULL, training = NULL,
  gauss = FALSE, dfupdate = "approx", eps = c(0.001, 0.1),
  verbose = TRUE, maxit = c(Inf, Inf), convstyle = "aitkens",
  parallel.cores = FALSE, ememargs = list(25, 5, "UUUU", "hard")))
```

and takes the following arguments:

- `x`: A numeric matrix, data frame, or vector (for univariate data).
- `Gs`: A number or vector indicating the number of groups to fit. Default is 1–9.
- `models`: A character vector specifying the models to fit. Models can be chosen using the terminology from Tables 1 and 2. Alternatively, notation from the popular **mclust** package can be used, using `"V"` for variable and `"E"` for equal across groups. In this case, the first letter refers to volume, the second to shape, the third to orientation, and fourth to degrees of freedom. Furthermore, the user can specify common groups of models such as `"all"`, `"dfconstrained"`, `"dfunconstrained"`, `"univariate"`, and `"gaussian"`. When `"gaussian"` is specified, the 14 multivariate Gaussian equivalents are used.

- `init`: A list of initializing classifications of the form `init[[G]]` that contains the initializing vector for all G groups considered (see example in Section 4.4). Alternatively, the user can specify a character string indicating an initialization method. Currently, the user can choose from `"kmeans"` (default), `"hard"` random, `"soft"` random, `"uniform"`, and `"emem"`. See Section 2.3 for further details on these options.

- `scale`: Logical indicating whether or not the function should scale the data. Default is `TRUE` – note that **teigen** models are not scale invariant.

- `dfstart`: The initialized value for the degrees of freedom. The default is 50.

- `known`: A vector of known classifications that can be numeric or character – must be the same length as the number of rows in the data set. If using in a true classification sense, give samples with unknown classification the value `NA` within `known` (see Section 4.3 below).

- `training`: Optional indexing vector for the observations whose classification is taken to be known.

- `gauss`: Logical indicating if the algorithm should use the Gaussian distribution. If `models = "gaussian"` then `gauss = TRUE` is forced.

- `dfupdate`: Character string or logical indicating how the degrees of freedom should be estimated. The default is `"approx"`, indicating a closed-form approximation be used. Alternatively, `"numeric"` can be specified, which makes use of `uniroot()`. If `FALSE`, the value from `dfstart` is used and the degrees of freedom are not updated. If `TRUE`, `"numeric"` will be used for backward-compatibility.

- `eps`: Vector (of size 2) giving tolerance values for the convergence criterion. First value is the tolerance level for iterated CM-steps. Second value is tolerance for the EM algorithm: convergence is based on Aitken's acceleration (default) or lack of progress. See Sections 2.1 and 2.5 for relevant details.

- `verbose`: Logical indicating whether the running output discussed in Section 2.4 should be displayed. This option is not available in parallel. The output displayed depends on the width of the R window. With a width of 80 or larger: time run, estimated time remaining, and percent complete are all displayed.

- `maxit`: Vector (of size 2) giving maximum iteration number for the iterated CM-steps and EM algorithm, respectively. A warning is displayed if either of these maximums are met.

- `convstyle`: Character string specifying the method of determining convergence. Default is
`"aitkens"`, which uses the criterion based on Aitken's acceleration, but lack of progress `"lop"` may be specified instead.

- `parallel.cores`: Logical indicating whether to run `teigen` in parallel or not. If `TRUE`, then the function discerns the number of cores available and uses all of them. Alternatively, a positive integer may be provided indicating the number of cores the user wishes to use for running in parallel.

- `ememargs`: A list of the controls for the emEM initialization: `numstart` – number of starts (default 25); `iter` – number of EM iterations (default 5); `model` – character string for the model name to be used (default `"UUUU"` from the C, U, I nomenclature, see details below); `init` – character string for the initialization method for emEM (default `hard`, or `soft`, or `kmeans`). The emEM initialization will run multiple, randomized initialization attempts for a limited number of iterations, and then continue the model-fitting process.

Output from `teigen()` is an object of class `teigen`, which can be manipulated as a two-pronged list object. The main contents are the results from the model chosen by the BIC, with an additional list containing the results from the model chosen by the ICL:

- `x`: Data used for clustering/classification.
- `index`: Indexing vector giving observations taken to be known (only available when `clas` is set greater than 0 or `training` is given).
- `classification`: Vector of group classifications as determined by the BIC.
- `bic`: BIC of the best fitted model.
- `modelname`: Name of the best model according to the BIC.
- `allbic`: Matrix of BIC values according to model and G. A value of `-Inf` is returned when a model does not converge.
- `bestmodel`: Character string giving best model (BIC) details.
- `G`: Value corresponding to the number of components chosen by the BIC.
- `tab`: Classification table for BIC model (only available when `known` is given). When classification is used, the "known" observations are left out of the table.
- `fuzzy`: The fuzzy clustering matrix for the model selected by the BIC.
- `logl`: The log-likelihood corresponding to the model with the best BIC.
- `iter`: The number of iterations until convergence for the model with the best BIC.
- `parameters`: List containing the fitted parameters: `mean` – matrix of means where the rows correspond to the component and the columns are the variables; `sigma` – array of scale covariance matrices (multivariate) or scale variances (univariate); `lambda` – vector of scale parameters, or constants of proportionality; `d` – array of eigenvectors, or orientation matrices; `a` – array of diagonal matrices proportional to eigenvalues, or shape matrices; `df` – vector containing the degrees of freedom for each component; `weights` – matrix of the expected value of the characteristic weights; `pig` – a vector giving the mixing proportions.
- `iclresults`: List containing all the previous outputs, except `x` and `index`, pertaining to the model chosen by the best ICL (all under the same name except `allicl` and `icl` are the equivalent of `allbic` and `bic`, respectively).
- `info`: List containing a few of the original user inputs, for use by other dedicated methods of the `teigen` class.

3.2. `plot.teigen()`

The S3 `plot` method for objects of class `teigen` provides bivariate marginal contour and/or uncertainty plots – for univariate data, it provides a univariate density plot. The function has the following usage:

```
plot(x, xmarg = 1, ymarg = 2, res = 200, what = c("contour", "uncertainty"),
  alpha = 0.4, col = rainbow(x$G), pch = 21, cex = NULL, bg = NULL, lty = 1,
  uncmult = 0, levels = c(seq(0.01, 1, by = 0.025), 0.001), main=NULL,
  xlab=NULL, draw.legend=TRUE, ...)
```

- `x`: An object of class `teigen`.
- `xmarg`: Scalar argument giving the number of the variable to be used on the x-axis.
- `ymarg`: Scalar argument giving the number of the variable to be used on the y-axis. Can be set to `NULL` for a univariate marginal density of `xmarg` (using `x[, xmarg]` as data).
- `res`: Scalar argument giving the resolution for the calculation grid required for the contour plot. Default is 200, which results in a 200×200 grid. Also determines how smooth the univariate density curves are (higher `res`, smoother curves). Ignored for uncertainty plots.
- `what`: Only used if the model provided by `x` is multivariate, this argument is a character vector stating which plots should be sent to the graphics device – choices are `"contour"` or `"uncertainty"`. If not provided, an interactive prompt will appear and provide these options.
- `alpha`: A factor modifying the opacity for the plotted points. Typically provided on the interval $[0, 1]$.
- `col`: A specification for the default plotting color – see section 'Colour Specification' in the `par` documentation. Note that the number of colors provided must equal to the number of groups in the `teigen` object (extra colors ignored).
- `pch`: Either an integer specifying a symbol or a single character to be used as the default in plotting points – see `points` documentation for possible values and their interpretation. If `pch` is one of 21:25, see `bg` for coloring.
- `cex`: A numerical value specifying the amount by which plotting text and symbols should be magnified relative to the default. For uncertainty plots, `cex` changes the size of the smallest sized point. The relative sizes amongst the points remains the same. As a result, the sizes of all the points change.
- `bg`: Background (fill) color for the open plot symbols if `pch` is one of $21 : 25$. If no `bg` is provided to color the inside of the points, then `col` will be used instead.
- `lty`: The line type for univariate plotting. See `par` documentation for more information. Only updates the group curves, not the density or mixture curves.
- `uncmult`: A multiplier for the points on the uncertainty plot. The larger the number, the more the size difference becomes exaggerated.

- `levels`: Numeric vector giving the levels at which contours should be drawn. Default is to draw a contour in 0.25 steps, plus a contour at 0.001. This may result in more/less contours than desired depending on the resulting density.
- `main`: Optional character string for title of plot. Useful default if left as `NULL`.
- `xlab`: Optional character string for x-axis label.
- `draw.legend`: Logical for a default generation of a legend to the right of the plot.
- `...`: Options to be passed to `plot`.

Note that if `ymarg` is `NULL` or the model provided by `x` is univariate, then `plot()` will provide a univariate marginal density.

4. Examples

Herein, we present a number of clustering/classification examples to illustrate the **teigen** package. We make use of a number of data sets from the base R distribution, as well as a couple available in the **gclus** library (Hurley 2012). Note that all `teigen()` calls in this illustration will force `verbose = FALSE` as the output produced will be computer specific; the reader is encouraged to set this argument to `TRUE` (the default value) if desired. In the interest of reproducibility, we use the `set.seed()` function when appropriate.

4.1. Model-based clustering: geometric example

To begin, we will provide a simple bivariate simulation to illustrate the geometrical difference between some of the **teigen** models. First, we generate the data using **clusterGeneration** (Qiu and Joe 2015) – a package that can be used to simulate groups of data.

```
R> library("clusterGeneration")
R> set.seed(542687)
R> sim <- genRandomClust(2, sepVal = .35, numReplicate = 1,
+    outputDatFlag = FALSE, outputLogFlag = FALSE, outputEmpirical = FALSE,
+    outputInfo = FALSE)$datList[[1]]
```

The `genRandomClust` command above specifies simulating 2 groups, with a `sepVal` of 0.35 (well-separated groups) and `numReplicate` indicates that we only seek one data set for this illustration. The remaining arguments are used to avoid writing files to the user's working directory. Now we fit several **teigen** models to this simulation and plot the results in Figure 1.

```
R> library("teigen")
R> par(mfrow = c(2, 2))
R> set.seed(431)
R> CIIC <- teigen(sim, 2, models = "CIIC", verbose = FALSE)
R> plot(CIIC, levels = seq(0.03, 1, by = 0.1), what = "contour",
+    xlab = "Variable 1", ylab = "Variable 2", main = "CIIC model")
R> set.seed(431)
```

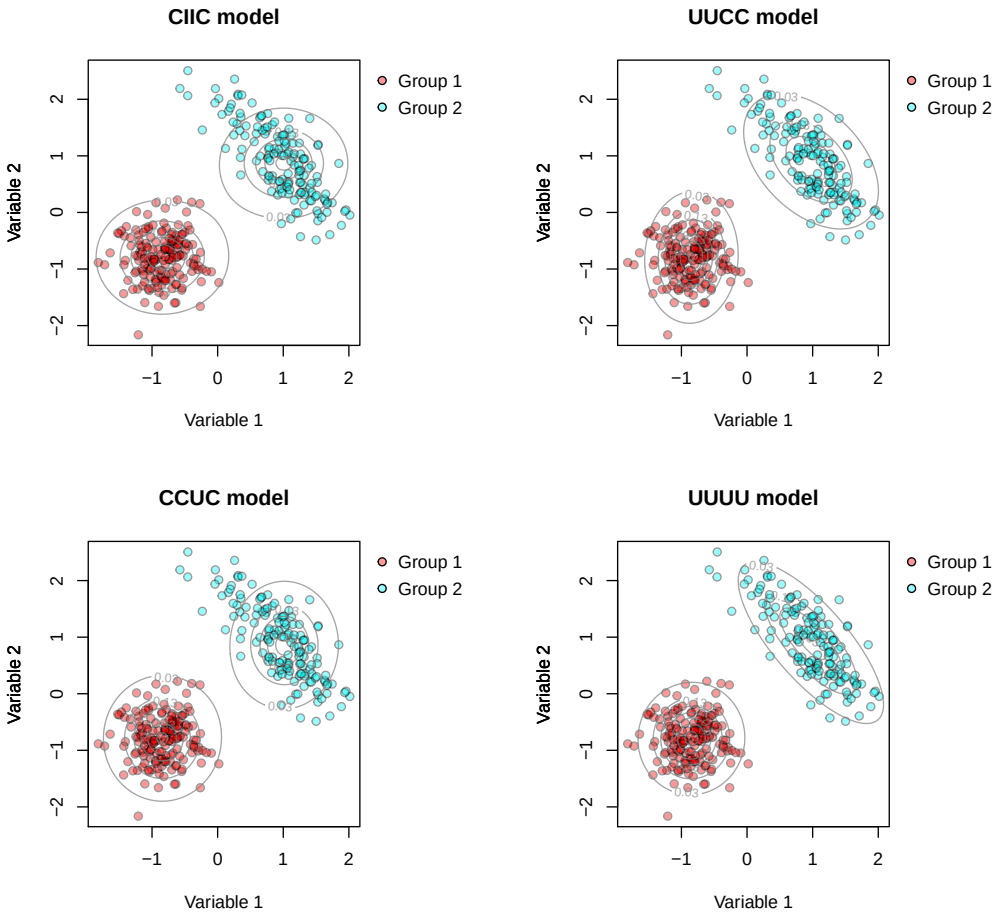

Figure 1: Contour plots produced using the the S3 method for plot on a `teigen` object. These four plots show some of the geometrical differences between the different tEIGEN models. The BIC values are −1361.1 (CIIC), −1255.7 (UUCC), −1357.4 (CCUC), and −1209.2 (UUUU).

```
R> UUCC <- teigen(sim, 2, models = "UUCC", verbose = FALSE)
R> plot(UUCC, levels = seq(0.03, 1, by = 0.1), what = "contour",
+    xlab = "Variable 1", ylab = "Variable 2", main = "UUCC model")
R> set.seed(431)
R> CCUC <- teigen(sim, 2, models = "CCUC", verbose = FALSE)
R> plot(CCUC, levels = seq(0.03, 1, by = 0.1), what = "contour",
+    xlab = "Variable 1", ylab = "Variable 2", main = "CCUC model")
R> set.seed(431)
R> UUUU <- teigen(sim, 2, models = "UUUU", verbose = FALSE)
R> plot(UUUU, levels = seq(0.03, 1, by = 0.1), what = "contour",
+    xlab = "Variable 1", ylab = "Variable 2", main = "UUUU model")
```

The constrained eigen-decomposition on the scale matrix of the multivariate t distribution has approximately the same effect, geometrically speaking, as this constraint would on the covariance matrix of the multivariate Gaussian distribution. Therefore, in the cases plotted in Figure 1, the CIIC model corresponds to 'spherical with equal volume', the UUCC model corresponds to 'ellipsoidal with equal shape', the CCUC model corresponds to 'ellipsoidal,

equal volume, and orientation', and the UUUU model allows for ellipsoids with 'varying volume, shape, and orientation' in addition to giving no constraint on the degrees of freedom.

4.2. Model-based clustering: The basics on Old Faithful

The Old Faithful data set is available in base R distributions as `faithful`. It is bivariate data measuring the time to eruption and length of eruption, both in minutes. In this example, we use the entire *t*EIGEN family initialized with soft random starting values.

```
R> library("teigen")
R> data("faithful")
R> set.seed(13786)
R> teigen_faith <- teigen(faithful, Gs = 1:4, init = "soft", scale = FALSE,
+    verbose = FALSE)
R> summary(teigen_faith)

------------- Summary for teigen -------------
            ------   RESULTS   ------
            Loglik:          -1130.182
            BIC:             -2327.635
            ICL:             -2328.299
            Model:           UUUC
            # Groups:        2

Clustering Table:

  1   2
 97 175

R> plot(teigen_faith, what = "contour")
```

The plot is shown in Figure 2. As noted in Section 2.3, using the `"soft"` initialization is not recommended due to the sensitivity of the EM algorithm. We can illustrate this sensitivity by re-running with a different seed.

```
R> set.seed(73)
R> teigen_faith2 <- teigen(faithful, Gs = 1:4, init = "soft", scale = FALSE,
+    verbose = FALSE)
R> summary(teigen_faith2)

------------- Summary for teigen -------------
          ---- BIC RESULTS ----
          Loglik:       -1126.506
          BIC:          -2320.282
          Model:        CCCC
          # Groups:     3
```

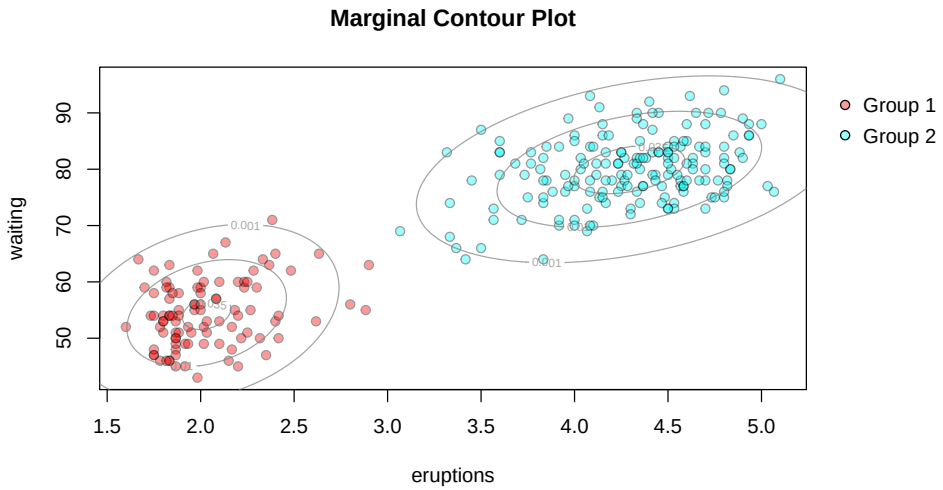

Figure 2: Contour plot of the Old Faithful data set produced by using the plot S3 method on a `teigen` object.

```
Clustering Table:

 1   2   3
47 128  97

        ---- ICL RESULTS ----
        Loglik:       -1130.19
        ICL:          -2328.266
        Model:        UUUC
        # Groups:     2


Clustering Table:

 1   2
97 175
```

It is worth noting that the ICL criterion still suggests the same two-group model. In order to illustrate the uncertainty plot, we plot the three-group solution in Figure 3 via

```
R> plot(teigen_faith2, what = "uncertainty")
```

With Figure 3, we can see the areas of high uncertainty in the classification results by focusing our attention on the larger points – the larger the point, the more uncertainty associated with its classification.

We can also illustrate the `predict.teigen` S3 method here by creating a new observation. By default, the BIC is used to select the model, but we can alternatively use the ICL via the argument `modelselect`.

```
R> predict(teigen_faith2, newdata = data.frame(eruptions = 2, waiting = 70))
```

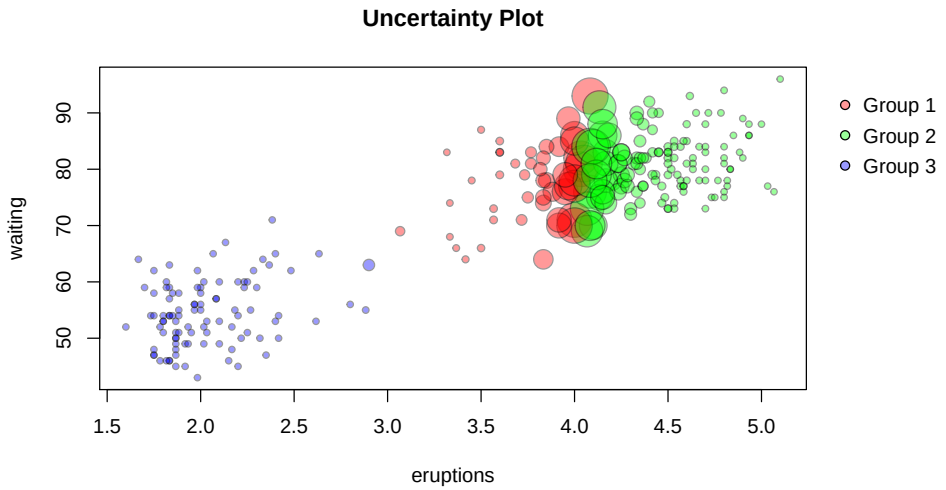

Figure 3: Uncertainty plot of the Old Faithful data set produced by using plot S3 method on a `teigen` object. Larger point sizes indicate a higher level of uncertainty regarding the classification of the observation.

```
$fuzzy
             [,1]          [,2]      [,3]
[1,] 1.085788e-07 9.018595e-13 0.9999999

$classification
[1] 3
```

```
R> predict(teigen_faith2, newdata = data.frame(eruptions = 2, waiting = 70),
+    modelselect = "ICL")

$fuzzy
          [,1]         [,2]
[1,] 0.9999414 5.859729e-05

$classification
[1] 1
```

Note that if the user specified that the data be scaled via `scale = TRUE` (which is default), then `predict.teigen` will take care of the scaling when predicting new values. As such, issues will arise if the user is not inputting `newdata` that is/are consistent with the original `teigen` call.

4.3. Semi-supervised model-based classification: the basics on iris

The famous iris data set is available in base R distributions as `iris`. It contains four measurements on 150 irises, hailing from three different species. We demonstrate the various ways of performing model-based classification with the **teigen** package. First, we use only the models from *t*EIGEN with unconstrained degrees of freedom. In a true classification scenario, there

exists a subset of the data where group membership is unknown. In this scenario, the known classification vector should have `NA` inputted for those values. The following example simulates these circumstances. We randomly take 50% of the data to have unknown membership and use the uniform initialization.

```
R> set.seed(357678)
R> irisknown <- iris[, 5]
R> irisknown[sample(1:nrow(iris), nrow(iris)/2)] <- NA
R> irisknown[1:5]

[1] setosa <NA>   setosa <NA>   <NA>
Levels: setosa versicolor virginica

R> tclass_iris3 <- teigen(iris[, -5], models = "dfunconstrained",
+    init = "uniform", known = irisknown, verbose = FALSE)
R> tclass_iris3$tab

            newmap
             setosa versicolor virginica
  setosa          0          0         0
  unknown        23         27        25
  versicolor      0          0         0
  virginica       0          0         0
```

We can see now that of the observations with unknown membership, 23 are classified as the *Iris setosa* species, while 27 are classified as *Iris versicolor* and 25 as *Iris virginica*.

The index of the observations that are taken to be known are stored in `$index`. Therefore, as an illustration, we can run the exact same analysis as before, but this time using a training index rather than specifying unknown classifications in the `known` vector.

```
R> trainingset <- tclass_iris3$index
R> tclass_iris2 <- teigen(iris[, -5], models = "dfunconstrained",
+    init = "uniform", known = iris[, 5], training = trainingset,
+    verbose = FALSE)
R> tclass_iris2$tab

            newmap
             setosa versicolor virginica
  setosa         23          0         0
  versicolor      0         26         0
  virginica       0          1        25
```

Because we actually know the true classification of the irises, we can verify the accuracy of the algorithm this way. In this case, we have misclassified one of the 'unknown' observations as *Iris versicolor*, when in fact it is from the *Iris virginica* species.

The classification table above represents the results from one particular classification run, and therefore may not be indicative of **teigen**'s classification performance on the iris data

set. Therefore, we now fit **teigen** models using 100 different training sets with $\frac{2}{3}$ of the data considered to have known group membership. We emphasize that this analysis is still in a semi-supervised context, so care needs to be taken in the interpretation of 'training' versus 'testing' sets since all observations are used in the model fitting.

```
R> set.seed(4518)
R> des <- matrix(1:150, 150, 100)
R> des <- apply(des, 2, function(v) v[-c(sample.int(150, 50,
+    replace = FALSE))])
R> results_list <- apply(des, 2, function(v) teigen(iris[, -5],
+    init = "uniform", known = iris[, 5], training = v, verbose = FALSE))
R> table_list <- lapply(results_list, function(v) v$tab)
R> misclass <- unlist(lapply(table_list, function(v) 1 - sum(diag(v))/50))
R> mean(misclass)

[1] 0.031

R> sd(misclass)

[1] 0.02076808
```

We can see an average misclassification rate of 3.1%, with a standard deviation of around 2.1%. We can also aggregate the classification tables from all of the observations.

```
R> Reduce("+", table_list)

            newmap
             setosa versicolor virginica
  setosa       1683          0         0
  versicolor      0       1530       116
  virginica       0         39      1632
```

This shows us, for instance, that the *Iris setosa* species is never misclassified through the 100 runs.

4.4. Model-based clustering: custom initialization on wine

The wine data set is available in the **gclus** library as `wine`. It contains 13 chemical measurements on 178 samples of Italian red wine. Herein, we analyze the data using model-based clustering with the entire *t*EIGEN family and illustrate how to perform user-specified initializations. We initialize using hierarchical clustering via the `hclust` function in R. We also illustrate the usage of the ICL as model selection and show how to find model parameters.

```
R> library("gclus")
R> data("wine")
R> hwine <- hclust(dist(scale(wine[, -1])))
R> initial <- lapply(1:5, function(i) cutree(hwine, k = i))
R> teigen_wine <- teigen(wine[, -1], Gs = 1:5, init = initial,
+    verbose = FALSE)
R> summary(teigen_wine)
```

```
------------- Summary for teigen -------------
         ---- BIC RESULTS ----
         Loglik:      -2517.76
         BIC:         -5444.88
         Model:       CIUC
         # Groups:    3

Clustering Table:

 1  2  3
65 61 52

         ---- ICL RESULTS ----
         Loglik:      -2368.811
         ICL:         -5449.59
         Model:       UCCU
         # Groups:    3

Clustering Table:

 1  2  3
59 71 48
```

Note that in the summary above, the distinct results are provided for the BIC and ICL. When this happens while using the **teigen** package, it is an indication that the two selection criteria disagree on the 'best' model. The user can discern pertinent information from the model chosen by the ICL through the `$iclresults` part of the **teigen** object. As an example, we can take a look at the degrees of freedom and the classification table from the ICL chosen model.

```
R> teigen_wine$iclresults$parameters$df

[1] 45.352618  8.270961 25.034678

R> table(wine[,1], teigen_wine$iclresults$class)

     1  2  3
  1 59  0  0
  2  0 71  0
  3  0  0 48
```

In this case, the *t*EIGEN family performs perfect clustering, correctly grouping all red wines by varietal when using `hclust` as an initialization method and the ICL as the selection criteria. We can illustrate, however, the sensitivity to starting values by performing the same analysis with k-means as the initialization method.

```
R> teigen_wine_k <- teigen(wine[, -1], Gs = 1:5, verbose = FALSE)
R> summary(teigen_wine_k)
```

```
------------- Summary for teigen -------------
            ------   RESULTS   ------
            Loglik:          -2435.972
            BIC:             -5431.578
            ICL:             -5438.901
            Model:           UIUC
            # Groups:        4

Clustering Table:

 1  2  3  4
29 50 46 53

R> table(wine[,1], teigen_wine_k$iclresults$class)

     1  2  3  4
  1  9 50  0  0
  2 20  0 46  5
  3  0  0  0 48
```

4.5. Model-based clustering: emEM initialization on ckd

In the **teigen** package, we have included a cleaned-up version of the chronic kidney disease (`ckd`) data set that can be found in the UCI Machine Learning Repository (Lichman 2013). In this version, we have removed the categorical variables and any observations with missing values on the remaining variables. After these adjustments, the data set contains 203 patients with 11 diagnostic measures each and the classifying variable indicating whether or not they have chronic kidney disease.

We now apply `teigen` to this data set while using the `"emem"` initialization (described in Section 2.3).

```
R> data("ckd")
R> set.seed(9798)
R> teigen_ckd <- teigen(ckd[,-1], init = "emem",
+    ememargs = list(numstart = 100, iter = 5, model = "UUUU",
+    init = "soft"), verbose=FALSE)
R> teigen_ckd

BIC and ICL select the same model and groups.
The best model (BIC of -4085.98, ICL of -4091.5653) is UIUU with G=3
```

We can compare this to a similar fitting which uses only one instance of a soft random initialization.

```
R> set.seed(9798)
R> teigen_ckd2 <- teigen(ckd[,-1], init = "soft", verbose=FALSE)
R> teigen_ckd2
```

```
BIC and ICL select the same model and groups.
The best model (BIC of -4102.76, ICL of -4110.4846) is UIUU with G=4
```

As we can see, the more thorough `emem` initialization results in a better model fit (BIC of -4086 versus -4103) when compared to a single random initialization.

4.6. Model-based clustering: univariate clustering on bank

The bank notes data set is available in the **gclus** library as `bank`. It contains a number of measurements on both counterfeit and genuine Swiss bank notes. We select the `Diagonal` variable to perform univariate clustering on.

```
R> library("gclus")
R> data("bank")
R> attach(bank)
R> set.seed(20637)
R> teigen_bank <- teigen(Diagonal, init = "hard", verbose = FALSE,
+    scale = FALSE)
R> summary(teigen_bank)

------------- Summary for teigen -------------
            ------   RESULTS   ------
            Loglik:         -268.2134
            BIC:            -568.2166
            ICL:            -576.7961
            Model:          univUC
            # Groups:       2

Clustering Table:

  1   2
104  96

R> table(Status, teigen_bank$class)

Status   1   2
     0   4  96
     1 100   0
```

The number of misclassified bank notes is only four. Interestingly, these four bank notes are all true notes that have been misclassified as counterfeit. Here we briefly illustrate the plotting function for univariate data (see Figure 4).

```
R> plot(teigen_bank, xlab = "Diagonal (in mm)")
```

Note that the `res` argument determines the number of values evaluated for the internal `curve()` call for a univariate plot; thus it, again, affects resolution (see Figure 5).

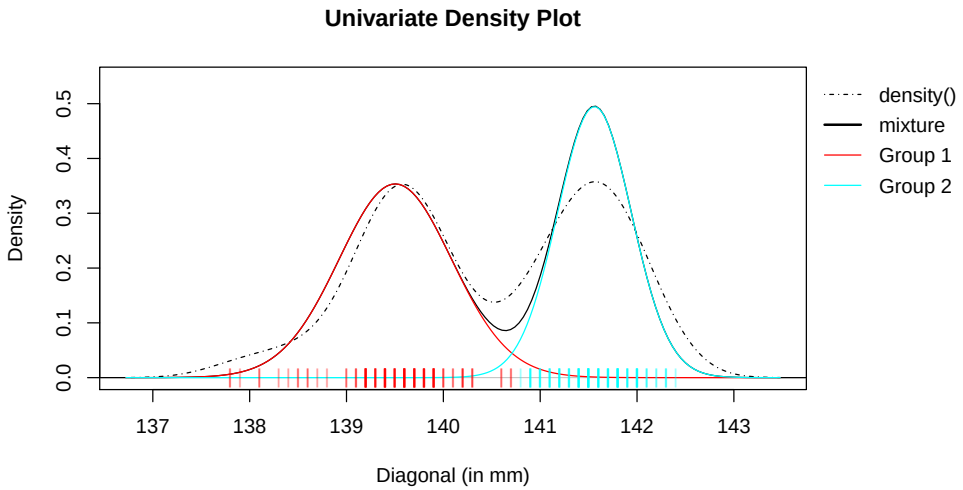

Figure 4: Density plot of the diagonal measure from the bank data set produced by using the plot S3 method on a univariate `teigen` object.

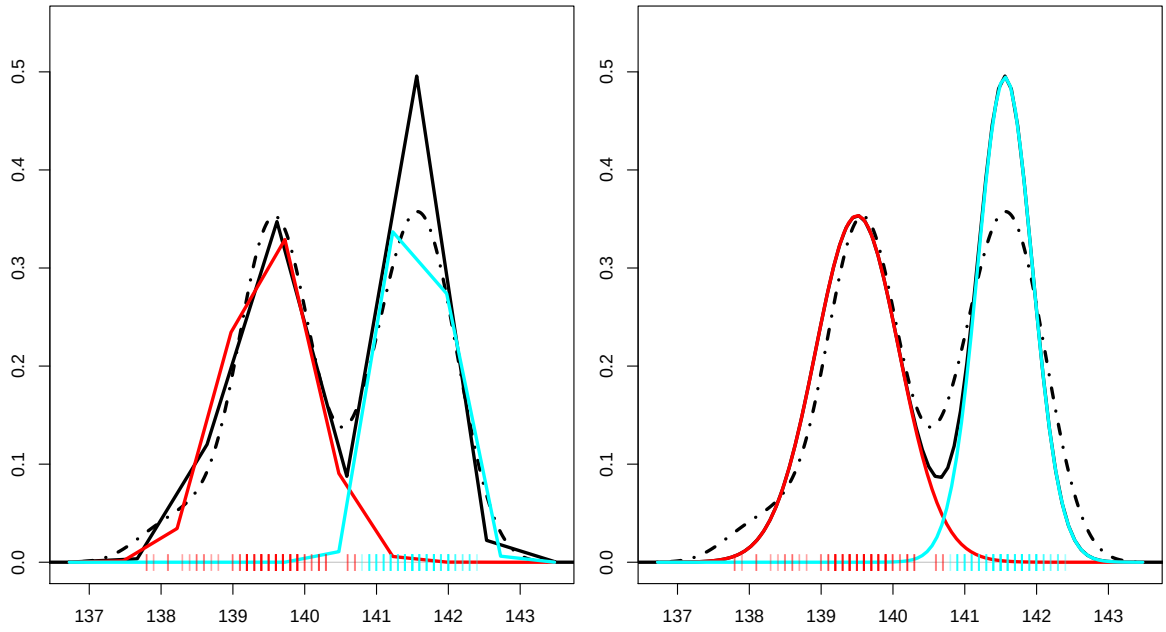

Figure 5: Density plots of the diagonal measure from the bank data set produced by using the plot S3 method on a univariate `teigen` object with differing resolutions.

```
R> par(mfrow = c(1, 2))
R> par(mar = c(2, 2, 1, 1))
R> plot(teigen_bank, res = 10, draw.legend = FALSE, main = "", lwd = 3)
R> plot(teigen_bank, res = 100, draw.legend = FALSE, main = "", lwd = 3)
```

If the resolution is too small, we can see cases like the left-hand side of Figure 5 where the density function is not evaluated close enough to the modes to provide an accurate representation.

4.7. Model-based clustering on a multi-core CPU

The parallelized method is now illustrated on the `body` data set from the **gclus** library. It contains a variety of physical measurements on male and female participants. Once again, we initialize via hierarchical clustering. The machine used for the demo analysis contains an octo-core CPU clocked at 3.5 GHz per core (AMD Phenom FX-8350). Note that if `parallel.cores = TRUE` then the `detectCores()` function from the **parallel** package will detect the number of cores available and use all of them.

```
R> library("gclus")
R> data("body")
R> bodydist <- dist(scale(body[, -25]))
R> hbody <- hclust(bodydist)
R> initial <- lapply(1:9, function(i) cutree(hbody, k = i))
R> system.time(teigen_body <- teigen(body[, -25], init = initial,
+    verbose = FALSE))

   user  system elapsed
 21.028   0.012  21.048

R> system.time(teigen_bodyp4 <- teigen(body[, -25], init = initial,
+    parallel.cores = 4))

   user  system elapsed
  0.480   0.184   9.100

R> system.time(teigen_bodyp8 <- teigen(body[, -25], init = initial,
+    parallel.cores = TRUE))

   user  system elapsed
  0.400   0.148   6.069
```

Comparing the time requirements of each model-fitting, we can see that speed improvements can be achieved by using several cores. We now check to see if the three runs are identical, and then check the clustering results.

```
R> identical(teigen_body, teigen_bodyp8)

[1] TRUE

R> identical(teigen_body, teigen_bodyp4)

[1] TRUE

R> table(body[,"Gender"], teigen_bodyp8$class)
```

```
    1   2
0 257   3
1   4 243
```

As we can see, when using the hierarchical clustering for initialization and the BIC as model-selection, there are seven misclassifications on the `body` data set.

4.8. Model-based clustering: Comparison with mclust on body

In the final example, we compare **teigen** clustering results with those of a popular multivariate Gaussian mixture model package that contains the equivalent eigen-decomposed covariance structure: version 5.2 of the **mclust** R package (Fraley *et al.* 2016). Care must be taken to make the comparison valid, so first the `body` data set from the **gclus** package (Hurley 2012) is loaded and scaled. This data set contains various physical and demographic measures on 507 adults. The intended grouping variable `Gender` is recorded in column 25 and is thus removed.

```
R> library(gclus)
R> library(mclust)
R> data(body)
R> sdata <- scale(body[, -25])
```

Next, the same initialization that `Mclust` makes use of must be supplied to the `teigen` call, so a list is supplied with the `hcVVV` results. We note here that we reduce the number of groups considered from the default for both methods (1–9) to 1–4. We assure the reader that neither method selects a larger number of groups than 4, so this is only to reduce computation time.

```
R> mclustinit <- list()
R> hcfit <- hcVVV(data = sdata)
R> for(i in 1:4) {
+   mclustinit[[i]] <- hclass(hcfit, i)
+ }
```

Finally, the convergence criteria and tolerance levels for both the M-step and the EM cycles need to be matched up. We specify the same tolerance levels and convergence criteria in `teigen` as the defaults for `Mclust`.

```
R> fitt <- teigen(sdata, Gs = 1:4, init = mclustinit, convstyle = "lop",
+    eps = c(sqrt(.Machine$double.eps), 1.e-5), verbose = FALSE)
R> fitg <- Mclust(sdata, G = 1:4, initialization = list(hcfit))
R> fitt

The best model (BIC of -18655.17) is UCCC with G=2
The best model (ICL of -18661.26) is UCCU with G=2

R> fitg

'Mclust' model object:
 best model: ellipsoidal, equal shape and orientation (VEE) with 4 components
```

```
R> table(body[, 25], fitt$classification)

     1   2
  0 257   3
  1   5 242

R> table(body[, 25], fitg$classification)

     1   2   3   4
  0  63   0   5 192
  1   2 187  57   1
```

As can be seen above, both models result in a misclassification of eight individuals. However, the results from the Gaussian model split each group up whereas the more robust **teigen** model retains the true group structure.

5. Summary

The *t*EIGEN family was developed further by introducing the four remaining multivariate mixture models based on an eigendecomposition of the scale matrix, as well as four univariate mixture models, and including a novel closed-form approximation for the degrees of freedom. Coinciding with this progress, the **teigen** R package was introduced and described in detail. It allows the user to fit the *t*EIGEN family of multivariate and univariate t distribution mixture models to numeric data in serial or parallel, with a number of built-in options and error-catches. Examples further illustrated how **teigen** can be used and showed its effectiveness as a clustering technique.

Acknowledgments

The authors gratefully acknowledge helpful comments on earlier versions of the **teigen** package from a number of colleagues, as well as the constructive critiques on both the software and manuscript from anonymous reviewers. This work was supported at various stages by a doctoral postgraduate scholarship (Andrews) from the Natural Sciences and Engineering Research Council of Canada (NSERC), individual NSERC Discovery Grants (Andrews, McNicholas), the Canada Research Chairs program (McNicholas) and the Research, Scholarly Activity, and Creative Achievement Fund at MacEwan University (Andrews). This document was produced using Sweave (Leisch 2002).

References

Aitken AC (1926). "A Series Formula for the Roots of Algebraic and Transcendental Equations." *Proceedings of the Royal Society of Edinburgh*, **45**, 14–22. doi:10.1017/s0370164600024871.

Andrews JL, McNicholas PD (2011a). "Extending Mixtures of Multivariate t Factor Analyzers." *Statistics and Computing*, **21**(3), 361–373. doi:10.1007/s11222-010-9175-2.

Andrews JL, McNicholas PD (2011b). "Mixtures of Modified t Factor Analyzers for Model-Based Clustering, Classification, and Discriminant Analysis." *Journal of Statistical Planning and Inference*, **141**(4), 1479–1486. doi:10.1016/j.jspi.2010.10.014.

Andrews JL, McNicholas PD (2012). "Model-Based Clustering, Classification, and Discriminant Analysis via Mixtures of Multivariate t Distributions." *Statistics and Computing*, **22**(5), 1021–1029. doi:10.1007/s11222-011-9272-x.

Andrews JL, McNicholas PD, Subedi S (2011). "Model-Based Classification via Mixtures of Multivariate t Distributions." *Computational Statistics & Data Analysis*, **55**(1), 520–529. doi:10.1016/j.csda.2010.05.019.

Andrews JL, Wickins JR, Boers NM, McNicholas PD (2018). ***teigen**: Model-Based Clustering and Classification with the Multivariate t Distribution*. R package version 2.2.2, URL https://CRAN.R-project.org/package=teigen.

Auder B, Lebret R, Iovleff S, Langrognet F (2016). ***Rmixmod**: An Interface for MIXMOD*. R package version 2.1.1, URL https://CRAN.R-project.org/package=Rmixmod.

Banfield JD, Raftery AE (1993). "Model-Based Gaussian and Non-Gaussian Clustering." *Biometrics*, **49**(3), 803–821. doi:10.2307/2532201.

Biernacki C, Celeux G, Govaert G (2000). "Assessing a Mixture Model for Clustering with the Integrated Completed Likelihood." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **22**(7), 719–725. doi:10.1109/34.865189.

Biernacki C, Celeux G, Govaert G (2003). "Choosing Starting Values for the EM Algorithm for Getting the Highest Likelihood in Multivariate Gaussian Mixture Models." *Computational Statistics & Data Analysis*, **41**(3), 561–575. doi:10.1016/s0167-9473(02)00163-9.

Böhning D, Dietz E, Schaub R, Schlattmann P, Lindsay B (1994). "The Distribution of the Likelihood Ratio for Mixtures of Densities from the One-Parameter Exponential Family." *Annals of the Institute of Statistical Mathematics*, **46**, 373–388. doi:10.1007/bf01720593.

Brent R (1973). *Algorithms for Minimization without Derivatives*. Prentice-Hall, New Jersey.

Browne RP, McNicholas PD (2014). "Estimating Common Principal Components in High Dimensions." *Advances in Data Analysis and Classification*, **8**(2), 217–226. doi:10.1007/s11634-013-0139-1.

Browne RP, McNicholas PD (2015). "A Mixture of Generalized Hyperbolic Distributions." *Canadian Journal of Statistics*, **43**(2), 176–198. doi:10.1002/cjs.11246.

Browne RP, McNicholas PD (2016). ***mixture**: Mixture Models for Clustering and Classification*. R package version 1.4, URL https://CRAN.R-project.org/package=mixture.

Celeux G, Govaert G (1995). "Gaussian Parsimonious Clustering Models." *Pattern Recognition*, **28**, 781–793. doi:10.1016/0031-3203(94)00125-6.

Dempster AP, Laird NM, Rubin DB (1977). "Maximum Likelihood from Incomplete Data via the EM Algorithm." *Journal of the Royal Statistical Society B*, **39**(1), 1–38. doi:10.1142/9789812388759_0028.

Fraley C, Raftery A, Scrucca L, Murphy TB, Fop M (2016). ***mclust**: Normal Mixture Modelling for Model-Based Clustering, Classification, and Density Estimation*. R package version 5.2, URL https://CRAN.R-project.org/package=mclust.

Fraley C, Raftery AE (1998). "How Many Clusters? Which Clustering Methods? Answers via Model-Based Cluster Analysis." *The Computer Journal*, **41**(8), 578–588. doi:10.1093/comjnl/41.8.578.

Fraley C, Raftery AE (2002). "Model-Based Clustering, Discriminant Analysis, and Density Estimation." *Journal of the American Statistical Association*, **97**(458), 611–631. doi:10.1198/016214502760047131.

Fraley C, Raftery AE, Murphy TB, Scrucca L (2012). ***mclust** Version 4 for R: Normal Mixture Modeling for Model-Based Clustering, Classification, and Density Estimation*. Technical Report 597, Department of Statistics, University of Washington.

Franczak BC, Browne RP, McNicholas PD (2014). "Mixtures of Shifted Asymmetric Laplace Distributions." *IEEE Transactions on Pattern Analysis and Machine Intelligence,*, **36**(6), 1149–1157. doi:10.1109/tpami.2013.216.

Hurley C (2012). ***gclus**: Clustering Graphics*. R package version 1.3.1, URL https://CRAN.R-project.org/package=gclus.

Karlis D, Santourian A (2009). "Model-Based Clustering with Non-Elliptically Contoured Distributions." *Statistics and Computing*, **19**, 73–83. doi:10.1007/s11222-008-9072-0.

Lee SX, McLachlan GJ (2013). "**EMMIXuskew**: An R Package for Fitting Mixtures of Multivariate Skew t Distributions via the EM Algorithm." *Journal of Statistical Software*, **55**(12), 1–22. doi:10.18637/jss.v055.i12.

Leisch F (2002). "`Sweave`: Dynamic Generation of Statistical Reports Using Literate Data Analysis." In *COMPSTAT 2002 — Proceedings in Computational Statistics*, pp. 575–580. Springer-Verlag.

Lichman M (2013). "UCI Machine Learning Repository." URL http://archive.ics.uci.edu/ml.

Lin TI (2010). "Robust Mixture Modeling Using Multivariate Skew t Distributions." *Statistics and Computing*, **20**, 343–356. doi:10.1007/s11222-009-9128-9.

Lindsay BG (1995). "Mixture Models: Theory, Geometry and Applications." In *NSF-CBMS Regional Conference Series in Probability and Statistics*, volume 5. Hayward, California: Institute of Mathematical Statistics.

McLachlan GJ, Peel D (1998). "Robust Cluster Analysis via Mixtures of Multivariate t Distributions." In *Lecture Notes in Computer Science*, volume 1451, pp. 658–666. Springer-Verlag, Berlin.

McNicholas PD (2016). *Mixture Model-Based Classification*. Chapman and Hall/CRC, Boca Raton.

McNicholas PD, Murphy TB (2008). "Parsimonious Gaussian Mixture Models." *Statistics and Computing*, **18**, 285–296. doi:10.1007/s11222-008-9056-0.

McNicholas PD, Murphy TB (2010). "Model-Based Clustering of Longitudinal Data." *The Canadian Journal of Statistics*, **38**(1), 153–168. doi:10.1002/cjs.10047.

Melnykov V, Melnykov I, Michael S (2015). "Semi-Supervised Model-Based Clustering with Positive and Negative Constraints." *Advances in Data Analysis and Classification*, pp. 1–23. doi:10.1007/s11634-015-0200-3.

Meng XL, Rubin DB (1993). "Maximum Likelihood Estimation via the ECM Algorithm: A General Framework." *Biometrika*, **80**, 267–278. doi:10.1093/biomet/80.2.267.

Qiu W, Joe H (2015). ***clusterGeneration**: Random Cluster Generation (with Specified Degree of Separation)*. R package version 1.3.4, URL https://CRAN.R-project.org/package=clusterGeneration.

R Core Team (2017). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

Schwarz G (1978). "Estimating the Dimension of a Model." *The Annals of Statistics*, **6**(2), 461–464. doi:10.1214/aos/1176344136.

Tortora C, Browne RP, Franczak BC, McNicholas PD (2015). ***MixGHD**: Model Based Clustering, Classification and Discriminant Analysis Using the Mixture of Generalized Hyperbolic Distributions*. R package version 1.8, URL https://CRAN.R-project.org/package=MixGHD.

Vrbik I, McNicholas PD (2012). "Analytic Calculations for the EM Algorithm for Multivariate Skew-Mixture Models." *Statistics & Probability Letters*, **82**(6), 1169–1174. doi:10.1016/j.spl.2012.02.020.

Vrbik I, McNicholas PD (2014). "Parsimonious Skew Mixture Models for Model-Based Clustering and Classification." *Computational Statistics & Data Analysis*, **71**, 196–210. doi:10.1016/j.csda.2013.07.008.

Vrbik I, McNicholas PD (2015). "Fractionally-Supervised Classification." *Journal of Classification*, **32**(3), 359–381. doi:10.1007/s00357-015-9188-9.

Wang K, Ng A, McLachlan G (2017). ***EMMIXskew**: The EM Algorithm and Skew Mixture Distribution*. R package version 1.0.2, URL https://CRAN.R-project.org/package=EMMIXskew.

A. Closed-form degrees of freedom

Here we provide further details and justification for the approximation for the degrees of freedom introduced in Section 2.2. Rearranging Equation 2, we get

$$\log\left(\frac{\hat{\nu}}{2}\right) - \varphi\left(\frac{\hat{\nu}}{2}\right) = -1 - \frac{1}{n}\sum_{g=1}^{G}\sum_{i=1}^{n}\hat{z}_{ig}(\log\hat{w}_{ig} - \hat{w}_{ig}) - \varphi\left(\frac{\hat{\nu}^{\text{old}}+p}{2}\right) + \log\left(\frac{\hat{\nu}^{\text{old}}+p}{2}\right).$$

The right-hand side is constant with respect to the newest estimate of $\hat{\nu}$; for ease of what follows, we define this as k. Taking the exponential, we are given

$$\frac{\hat{\nu}}{2\exp\left(\varphi\left(\frac{\hat{\nu}}{2}\right)\right)} = \exp(k).$$

Now, $\exp\left(\varphi\left(\frac{\hat{\nu}}{2}\right)\right)$ may be approximated by $\frac{\hat{\nu}}{2} - \frac{1}{2}$ for $\hat{\nu} > 2$. We plot the difference for this approximation for the interval (2,200] in Figure 6.

```
R> curve(exp(digamma(x/2)) - (x/2 - 1/2), from = 2.0001, to = 200)
```

The resulting plot shows a monotone decreasing error as the degrees of freedom increases, with a maximum value of approximately 0.0615 at $x = 2.0001$ and a minimum value of approximately 0.0004 at $x = 200$.

Inputting this approximation and solving for $\hat{\nu}$, we get

$$\hat{\nu} \approx \frac{-\exp(k)}{1-\exp(k)}. \tag{4}$$

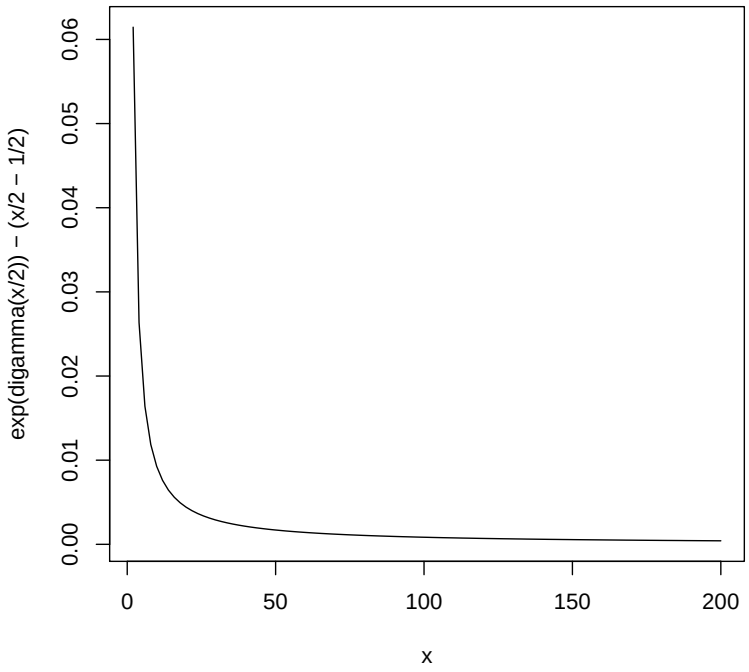

Figure 6: Plot of the difference between the exponential of a digamma function and its approximation for values between 2 and 200.

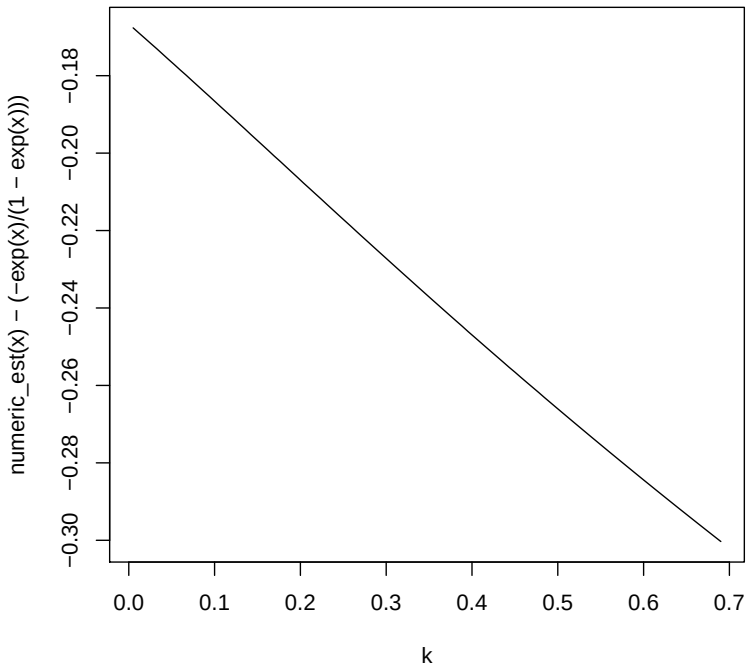

Figure 7: Plot of the difference between the numerically solved estimate and the closed form estimate for varying values of k.

In terms of estimates, we can check the differences between the numeric estimate and this approximation (plotting results in Figure 7).

```
R> numeric_est <- function(x) {
+   sapply(x, function(w) uniroot(function(j) - w + log(j/2) - digamma(j/2),
+     interval = c(1, 300))$root)
+ }
R> curve(numeric_est(x) - (-exp(x)/(1 - exp(x)) ), from = 0.0051, to = 0.69,
+    xlab = "k")
```

Along the x-axis in Figure 7 are the constants k for the calculations. As such, the estimate along the left-hand side is approximately close to 200 degrees of freedom, and the right is approximately 2 degrees of freedom. Clearly, the closed-form approximation is consistently smaller than the numeric estimate, and there is a linear relationship where the approximation is closer (within 0.17) at higher degrees of freedom. Because the error is relatively consistent (falls between 0.168 and 0.300 for all relevant estimates), we introduce a simple corrective term that makes use of the previous estimate:

$$\hat{\nu} \approx \frac{-\exp(k) + 2\exp(k)\left(\exp\left(\varphi\left(\frac{\hat{\nu}^{\text{old}}}{2}\right)\right) - \left(\frac{\hat{\nu}^{\text{old}}}{2} - \frac{1}{2}\right)\right)}{1 - \exp(k)}. \tag{5}$$

Finally, we note if we alternatively define

$$k = -1 - \frac{1}{n_g}\sum_{i=1}^{n} \hat{z}_{ig}(\log \hat{w}_{ig} - \hat{w}_{ig}) - \varphi\left(\frac{\hat{\nu_g}^{\text{old}} + p}{2}\right) + \log\left(\frac{\hat{\nu_g}^{\text{old}} + p}{2}\right)$$

then Equations 4 and 5 hold for $\hat{\nu}_g$ (unconstrained degrees of freedom) instead.

Interestingly, testing through simulations has shown that the numeric and closed estimations are essentially equivalent in terms of estimation accuracy. While the closed approximation tends to require more iterations than the numeric estimate before converging, it conversely takes less time per iteration – which means that in all cases we have noted a faster runtime for the end user when using the closed-form approximation. See discussion of `dfupdate` in Section 3 for details on setting this option.

Affiliation:

Jeffrey L. Andrews
Department of Statistics
University of British Columbia – Okanagan Campus
Kelowna, British Columbia, Canada
E-mail: jeff.andrews@ubc.ca
URL: http://stat.ok.ubc.ca/faculty/andrews.html

Jaymeson R. Wickins
MacEwan University
Edmonton, Alberta, Canada

Nicholas M. Boers
Department of Computer Science
MacEwan University
Edmonton, Alberta, Canada
E-mail: boersn@macewan.ca

Paul D. McNicholas
Department of Mathematics & Statistics
McMaster University
Hamilton, Ontario, Canada
E-mail: mcnicholas@math.mcmaster.ca
URL: http://www.paulmcnicholas.info/

Journal of Statistical Software http://www.jstatsoft.org/
published by the Foundation for Open Access Statistics http://www.foastat.org/
February 2018, Volume 83, Issue 7 *Submitted:* 2015-12-29
doi:10.18637/jss.v083.i07 *Accepted:* 2016-12-21