upclass: An R Package for Updating Model-Based Classification Rules

Niamh Russell
University College Dublin

Laura Cribbin
University of Limerick

Thomas Brendan Murphy
University College Dublin

Abstract

Standard methods for classification use labeled data to establish criteria for assigning unlabeled data to groups. However, the unlabeled data which need to be classified often contain important information about the structure of the groups, despite the group membership of these observations being unknown. A new R package called **upclass** is presented which uses both labeled and unlabeled data to construct a model-based classification method. The method uses the EM algorithm to obtain maximum likelihood estimates of the model parameters and classifications for the unlabeled data. It can be shown to perform better than classical methods, particularly in cases where few observations are labeled.

Keywords: classification, EM algorithm, **mclust**, R, semi-supervised classification.

1. Introduction

Classification techniques are employed regularly in a wide variety of application areas. Examples include food science applications where studies are carried out to establish whether products are correctly labeled (e.g., Caetano, Üstün, Hennessy, Smeyers-Verbeke, Melssen, Downey, Buydens, and Vander Heyden 2007; Toher, Downey, and Murphy 2007, 2011); botanical investigations to identify rare plants (e.g., Pouteau, Meyer, Taputuarai, and Stoll 2012) and medical diagnostic applications to identify whether patients have a particular disease or condition (e.g., Fan, Murphy, Byrne, Brennan, Fitzpatrick, and Watson 2011). It is important to devise effective rules in order to reduce potential errors.

Classification methods require a labeled dataset so that the number of groups and the structure of groups can be inferred. The task is to classify any unlabeled observations into the correct groups. Traditionally, a classification rule is developed using the fully labeled data which can then be used to classify any new unlabeled data as it becomes available. Extensive reviews of classification methods include Ripley (1996) and McLachlan (1992).

Semi-supervised classification methods use both the labeled *and* unlabeled data to develop a classifier for the unlabeled observations. These methods exploit the idea that even though the group memberships of the unlabeled data are unknown, these data carry important information about the group parameters (e.g., McLachlan 1977; O'Neill 1978; Dean, Murphy, and Downey 2006; Chapelle, Schölkopf, and Zien 2006). These methods provide a framework for updating a classification rule using unlabeled observations, so that more accurate classifications can be obtained. A number of semi-supervised classification methods have been developed including model-based methods (e.g., Dean *et al.* 2006; McNicholas 2010; Murphy, Dean, and Raftery 2010; Toher *et al.* 2011) and machine learning methods (e.g., Joachims 1999; Wang, Chen, and Zhou 2012). Detailed reviews of semi-supervised classification include Chapelle *et al.* (2006) and Zhu and Goldberg (2009).

Herein, we present the R package **upclass** which implements the (semi-supervised) updated model-based classification method as developed in Dean *et al.* (2006). This method starts by estimating the unknown labels using a standard model-based classification method and then combines them with the labeled observations to form the complete-data. The EM algorithm is utilized where the parameters and estimated unknown labels are iteratively updated until convergence. This yields maximum likelihood estimates for the parameters in the model and estimates group membership for the unlabeled observations.

The methods for standard model-based classification and clustering are outlined in Section 2 and the algorithm for the updated method is described in more detail in Section 3.

In Section 4, we present a short review of other semi-supervised methods, and outline what our package adds to the existing functionality.

In Section 5, we will illustrate the use of each function in the **upclass** package by working through examples using the well known olive oil data set (Forina, Armanino, Lanteri, and Tiscornia 1983) . In Section 5.10, we give a short comparison of the effectiveness of the updated method versus a fully supervised method for a case where only a small proportion of the data is labeled.

The R package implementing the methodology described in this article is available from the Comprehensive R Archive Network at `http://CRAN.R-project.org/package=upclass`.

2. Model-based methods

Discriminant analysis is concerned with classifying data into predefined groups while clustering sets out to cluster data into a previously undefined number of groups or clusters. In this section, we will outline the model-based approach to clustering (Section 2.1) and discriminant analysis (Section 2.2). The updated classification method which will be developed in Section 3 uses a hybrid of the statistical ideas underlying model-based discriminant analysis and clustering.

2.1. Model-based clustering

Model-based clustering (Banfield and Raftery 1993; Fraley and Raftery 2002, 2007) as implemented in the **mclust** package (Fraley, Raftery, Murphy, and Scrucca 2012) is used for clustering data into groups, where the number of groups G is unknown.

Model-based clustering is formulated as follows, we assume that there are G clusters, where each cluster g arises with probability τ_g (where $\sum \tau_g = 1$ and each τ_g is non-negative) and data within each cluster follows a normal distribution with cluster specific mean μ_g and covariance $\mathbf{\Sigma}_g$. That is, the data are characterized by a finite mixture of normal distributions.

Hence, the density of each observation can be given by,

$$f(y) = \sum_{g=1}^{G} \tau_g \phi(y|\mu_g, \mathbf{\Sigma}_g), \tag{1}$$

where $\phi(\cdot)$ is a multivariate normal density.

It is worth noting that the assumption of multivariate normal distributed clusters implies that the clusters are elliptical in shape. Banfield and Raftery (1993) proposed that constraints be placed on the covariance matrices in such a way as to allow for different conformations for the elliptical clusters. A modified eigenvalue decomposition of $\mathbf{\Sigma}_g$ is used to implement these variations. This decomposition can be written as

$$\mathbf{\Sigma}_g = \lambda_g \mathbf{D}_g \mathbf{A}_g \mathbf{D}_g^T,$$

where λ_g	–	is a constant which controls the cluster volume
$\mathbf{D}_g$	–	is an orthogonal matrix of eigenvectors which control the orientation/direction of the clusters
$\mathbf{A}_g$	–	is a diagonal matrix, with entries proportional to the eigenvalues, which control the shape of the cluster.

We can restrict each part of the covariance $\mathbf{\Sigma}_g$ in different ways, resulting in fourteen different possible models (Biernacki, Celeux, Govaert, and Langrognet 2006). Throughout this paper, we will consider the ten covariance structures implemented in **mclust** (Fraley *et al.* 2012), as displayed in Figure 1 and Table 1. Each letter in the name of a model corresponds to the constraint placed on the volume, shape and orientation respectively. The constraint can be equal (E), variable (V) or identity (I). Consider, for example, the EEV model for the covariance. If data are fitted by this model, then each cluster has the same volume and the same shape but the orientation of each cluster is allowed to differ.

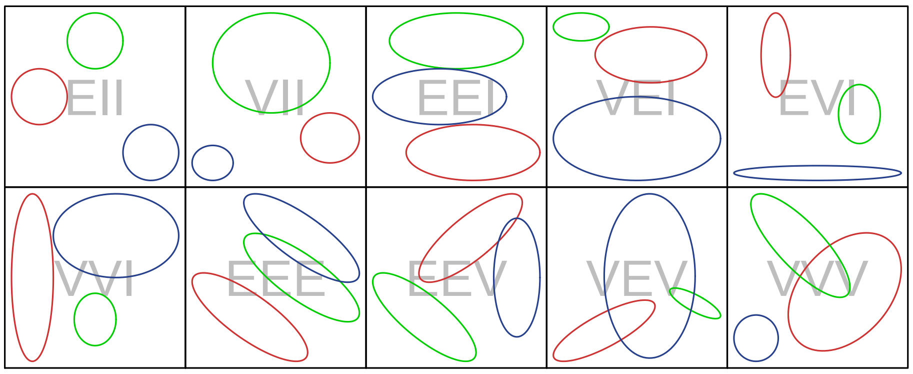


Figure 1: Examples of clusters under each covariance restriction.

As shown in Figure 1, the various covariance restrictions result in a different combination of cluster shapes in each model. The constraints yield parsimonious models which facilitate a more flexible modeling strategy beyond assuming unequal covariance (VVV) or equal covariance (EEE).

We will introduce some notation here, which will be used throughout. Let $(\mathbf{x}_N, \mathbf{l}_N)$ be the labeled data, where the observations are denoted by $\mathbf{x}_N = (x_1, x_2, \ldots, x_N)$ and their labels by $\mathbf{l}_N = (l_1, l_2, \ldots, l_N)$. The unlabeled data will be represented by $\mathbf{y}_M$ where $\mathbf{y}_M =$

Model	Volume	Shape	Orientation	Covariance $\mathbf{\Sigma}_g$
EII	Equal	Spherical		$\lambda\mathbf{I}$
VII	Variable	Spherical		$\lambda_g\mathbf{I}$
EEI	Equal	Equal	Axis aligned	$\lambda\mathbf{A}$
VEI	Variable	Equal	Axis aligned	$\lambda_g\mathbf{A}$
EVI	Equal	Variable	Axis aligned	$\lambda\mathbf{A}_g$
VVI	Variable	Variable	Axis aligned	$\lambda_g\mathbf{A}_g$
EEE	Equal	Equal	Equal	$\lambda\mathbf{DAD}^\top$
EEV	Equal	Equal	Variable	$\lambda\mathbf{D}_g\mathbf{AD}_g^\top$
VEV	Variable	Equal	Variable	$\lambda_g\mathbf{D}_g\mathbf{AD}_g^\top$
VVV	Variable	Variable	Variable	$\lambda_g\mathbf{D}_g\mathbf{A}_g\mathbf{D}_g^\top$

Table 1: Covariance decompositions available.

$(y_1, y_2, \dots, y_M)$ and the unknown labels are $\mathbf{z}_M = (z_1, z_2, \dots, z_M)$.
The model parameters are estimated using maximum likelihood via the EM algorithm (Dempster, Laird, and Rubin 1977). The EM algorithm, a technique used to find maximum likelihood estimates in cases where there are missing data, is made up of two steps, the Expectation (E) and the Maximization (M) steps. Since the data are from a mixture model, (See Equation 1) we can write the likelihood as the product over this density, evaluated at each y_m,

$$L(\tau, \mu, \mathbf{\Sigma}|\mathbf{y}_M) = \prod_{m=1}^{M} \left[\sum_{g=1}^{G} \tau_g \phi(y_m|\mu_g, \mathbf{\Sigma}_g) \right], \tag{2}$$

and the log-likelihood as

$$l(\tau, \mu, \mathbf{\Sigma}|\mathbf{y}_M) = \sum_{m=1}^{M} \log \left[\sum_{g=1}^{G} \tau_g \phi(y_m|\mu_g, \mathbf{\Sigma}_g) \right],$$

where $\tau = (\tau_1, \tau_2, \dots, \tau_G)$, $\mu = (\mu_1, \mu_2, \dots, \mu_G)$ and $\mathbf{\Sigma} = (\mathbf{\Sigma}_1, \mathbf{\Sigma}_2, \dots, \mathbf{\Sigma}_G)$.
It is difficult to maximize the log-likelihood directly because the log-likelihood is expressed as the log of a summation. To resolve this, we introduce indicator variables, z_{mg}, which represent the unknown labels of each observation,

$$\text{where} \quad z_{mg} = \begin{cases} 1 & \text{if } y_m \text{ is from group } g \\ 0 & \text{otherwise.} \end{cases}$$

The complete-data likelihood can now be written as

$$L(\tau, \mu, \mathbf{\Sigma}|\mathbf{y}_M, \mathbf{z}_M) = \prod_{m=1}^{M} \prod_{g=1}^{G} \left[\tau_g \phi(y_m|\mu_g, \mathbf{\Sigma}_g) \right]^{z_{mg}}, \tag{3}$$

and the complete-data log-likelihood is of the form

$$l(\tau, \mu, \mathbf{\Sigma}|\mathbf{y}_M, \mathbf{z}_M) = \sum_{m=1}^{M} \sum_{g=1}^{G} z_{mg} \left[\log \tau_g + \log \phi(y_m|\mu_g, \mathbf{\Sigma}_g) \right]. \tag{4}$$

The E-step of the algorithm replaces the z_{mg} values in Equation 4 with their conditional expected values, which are of the form

$$\hat{z}_{mg} = \frac{\hat{\tau}_g \phi(y_m|\hat{\mu}_g, \hat{\mathbf{\Sigma}}_g)}{\sum_{g'=1}^{G} \hat{\tau}_{g'} \phi(y_m|\hat{\mu}_{g'}, \hat{\mathbf{\Sigma}}_{g'})}, \text{ for all } m = 1, ..., M, \text{ and } g = 1, ..., G. \tag{5}$$

thus yielding the expected complete-data log-likelihood.

The M-step of the algorithm maximizes the expected complete-data log-likelihood function. The algorithm is iterated until convergence of the log-likelihood is reached and the final z_{mg} values provide the posterior probability that observation m belongs to group g.

Each observation is then classified to the group with maximum *a posteriori* (MAP) probability.

Further details on the EM algorithm are provided in Section 3, where the version of the algorithm for the updated method of model-based classification is described.

2.2. Model-based discriminant analysis

Two classical supervised classification methods are linear (LDA) and quadratic (QDA) discriminant analysis. Both methods can be seen as model-based discriminant analysis methods based on a similar model to that outlined in Section 2.1 but where the group membership for each observation is known. When implementing LDA, the covariance matrix is assumed equal for each group, which corresponds to the EEE covariance structure; whereas in QDA, each group is allowed to have its own unconstrained covariance matrix, which corresponds to the VVV covariance structure. It's possible, of course, to carry out supervised classification with all the Mclust models, as we will show in Sections 5.7 and 5.10.

In the context of supervised classification, there are two types of data: labeled data for which the group memberships are known and unlabeled data where they are unknown. Supervised classification uses labeled data to estimate model parameters which are used to create a classification rule. This rule can then be used to classify the unlabeled data.

Keeping the notation of Section 2.1, the likelihood of the labeled data can be written as:

$$L(\tau, \mu, \mathbf{\Sigma}|\mathbf{x}_N, \mathbf{l}_N) = \prod_{n=1}^{N} \prod_{g=1}^{G} [\tau_g \phi(x_n|\mu_g, \mathbf{\Sigma}_g)]^{l_{ng}}. \tag{6}$$

Hence, the log-likelihood is

$$l(\tau, \mu, \mathbf{\Sigma}|\mathbf{x}_N, \mathbf{l}_N) = \sum_{n=1}^{N} \sum_{g=1}^{G} l_{ng} [\log \tau_g + \log \phi(x_n|\mu_g, \mathbf{\Sigma}_g)].$$

$$\text{where } \; l_{ng} = \begin{cases} 1 & \text{if } x_n \text{ belongs to group } g \\ 0 & \text{otherwise.} \end{cases}$$

The function $l(\tau, \mu, \mathbf{\Sigma}|\mathbf{x}_N, \mathbf{l}_N)$ can be maximized with respect to $(\tau_g, \mu_g, \mathbf{\Sigma}_g)$ to obtain maximum likelihood estimates for the parameters $(\hat{\tau}_g, \hat{\mu}_g, \hat{\mathbf{\Sigma}}_g)$ in the model. Using these estimates, calculated from the labeled data, the expected value of the unknown labels $\mathbf{z}_M$ can be computed. They will have the same form as in Equation 5.

As before, the *a posteriori* probabilities of group membership for each observation can be used to derive the maximum *a posteriori* (MAP) predicted group membership for each observation.

2.3. Model selection

In implementing model-based discriminant analysis and clustering techniques, the model of the data must be known. If the model is not known, it is recommended to fit every model to the data and calculate the Bayesian Information Criterion (BIC) value (Schwarz 1978; Kass and Raftery 1995) for each model. The BIC value is calculated in such a way where it penalizes for a large number of parameters and rewards for a large likelihood value.

$$\text{BIC} = 2\log(L) - k\log(n)$$

where	L	–	is the likelihood of the data
	k	–	is the number of estimated model parameters
	n	–	is the number of observations.

The model with the highest BIC value is selected. While this does not always guarantee the lowest misclassification rate, in practice it often selects a close to optimal model (Biernacki and Govaert 1999). Biernacki and Govaert (1999) provide an overview of other model selection criteria for model-based clustering and classification.

3. The new updating method

The backbone of the method employed by **upclass** is the idea that the unlabeled data may potentially contain important information about the overall data even though their group memberships are unknown (Dean *et al.* 2006). This information can help give a clearer picture of the structure of the groups in the data. Earlier work in using labeled data to update model-based classification rules has been carried out by McLachlan (1975, 1977), Ganesalingam and McLachlan (1978) and O'Neill (1978) amongst others. More recent work includes Nigam, McCallum, and Mitchell (2006), McNicholas (2010) and Murphy *et al.* (2010).

We have observed $(\mathbf{x}_N, \mathbf{l}_N, \mathbf{y}_M)$ and unknown $\mathbf{z}_M$. So the observed likelihood is of the form

$$L(\tau, \mu, \boldsymbol{\Sigma}|\mathbf{x}_N, \mathbf{l}_N, \mathbf{y}_M) = \underbrace{\prod_{n=1}^{N}\prod_{g=1}^{G}[\tau_g\phi(x_n|\mu_g, \boldsymbol{\Sigma}_g)]^{l_{ng}}}_{\text{labeled data}} \underbrace{\prod_{m=1}^{M}\left[\sum_{g=1}^{G}\tau_g\phi(y_m|\mu_g, \boldsymbol{\Sigma}_g)\right]}_{\text{unlabeled data}}; \qquad (7)$$

this is the product of the likelihood for model-based discriminant analysis (Equation 6) and model-based clustering (Equation 2). If we treat the unknown labels as missing data, we can write the likelihood for the complete-data as

$$L_c(\tau, \mu, \boldsymbol{\Sigma}|\mathbf{x}_N, \mathbf{l}_N, \mathbf{y}_M, \mathbf{z}_M) = \underbrace{\prod_{n=1}^{N}\prod_{g=1}^{G}[\tau_g\phi(x_n|\mu_g, \boldsymbol{\Sigma}_g)]^{l_{ng}}}_{\text{labeled data}} \underbrace{\prod_{m=1}^{M}\prod_{g=1}^{G}[\tau_g\phi(y_m|\mu_g, \boldsymbol{\Sigma}_g)]^{z_{mg}}}_{\text{unlabeled data}}; \qquad (8)$$

this is a product of the likelihood for model-based discriminant analysis (Equation 6) and the complete-data likelihood for model-based clustering (Equation 3).

The package uses the complete-data log likelihood (Equation 8) and the EM algorithm to find maximum likelihood estimates for the unknown parameters of the model. The algorithm then uses these estimated parameters to classify the unlabeled data.

3.1. How it works

There are four general steps used to implement this updated classification rule. They iterate through the EM algorithm and are made up of the following:

Step 1 Let $k = 0$. Find initial values for the parameter estimates in the model. Only the labeled data $(\mathbf{x}_N, \mathbf{l}_N)$ are used here along with the M-step of the EM algorithm. This is equivalent to performing classical model-based discriminant analysis to obtain starting values for the model parameters, but within the structure of the **mclust** models.

Step 2 Using the current parameter estimates, $\hat{\tau}^{(k)}$, $\hat{\mu}^{(k)}$ and $\hat{\mathbf{\Sigma}}^{(k)}$, calculate the expected value of the unknown labels through the E-step,

$$\hat{z}_{mg}^{(k+1)} = \frac{\hat{\tau}_g^{(k)} \phi(y_m | \hat{\mu}_g^{(k)}, \hat{\mathbf{\Sigma}}_g^{(k)})}{\sum_{g'=1}^{G} \hat{\tau}_{g'}^{(k)} \phi(y_m | \hat{\mu}_{g'}^{(k)}, \hat{\mathbf{\Sigma}}_{g'}^{(k)})}.$$

Step 3 Combine $(\mathbf{x}_N, \mathbf{l}_N)$ and $(\mathbf{y}_M, \hat{\mathbf{z}}_M^{(k+1)})$ to form the complete-data. Using the complete-data, calculate new parameter estimates for the model, $\tau^{(k+1)}$, $\mu^{(k+1)}$ and $\mathbf{\Sigma}^{(k+1)}$, through the M-step by maximizing the complete-data log-likelihood (Equation 8).

Step 4 Check for convergence of the log-likelihood using the Aitken acceleration convergence criterion, by default. There is a simpler convergence option also, see Section 5.3. If convergence has been reached, stop. If not, set $k = k+1$ and return to Step 2 where new estimates for the unknown labels are calculated followed by new parameter estimates.

We chose to use the labeled data to calculate the initial parameter estimates. Toher *et al.* (2007) did some work on this and showed that, in most cases, this was the most effective method.

The parameter estimates used in Step 3 are of the following form. The estimate of $\hat{\tau}_g^{(k+1)}$ can be seen as the average number of observations in each group,

$$\hat{\tau}_g^{(k+1)} = \frac{\sum_{n=1}^{N} l_{ng} + \sum_{m=1}^{M} \hat{z}_{mg}^{(k+1)}}{N+M},$$

and $\hat{\mu}_g^{(k+1)}$ is a weighted average of the observations, where the labels and their estimates are used as weights,

$$\hat{\mu}_g^{(k+1)} = \frac{\sum_{n=1}^{N} l_{ng} \mathrm{x}_n + \sum_{m=1}^{M} \hat{z}_{mg}^{(k+1)} \mathrm{y}_m}{\sum_{n=1}^{N} l_{ng} + \sum_{m=1}^{M} \hat{z}_{mg}^{(k+1)}}.$$

The estimation of $\mathbf{\Sigma}_g$ depends on the constraints placed on the covariance matrix. For example, if the model was VVV, the estimate for $\hat{\mathbf{\Sigma}}_g^{(k+1)}$ would look like;

$$\hat{\mathbf{\Sigma}}_g^{(k+1)} = \frac{\sum_{n=1}^{N} l_{ng}(\mathbf{x}_n - \hat{\mu}_g^{(k+1)})(\mathbf{x}_n - \hat{\mu}_g^{(k+1)})' + \sum_{m=1}^{M} \hat{z}_{mg}^{(k)}(\mathbf{y}_m - \hat{\mu}_g^{(k+1)})(\mathbf{y}_m - \hat{\mu}_g^{(k+1)})'}{\sum_{n=1}^{N} l_{ng} + \sum_{m=1}^{M} \hat{z}_{mg}^{(k)}}.$$

For further details on parameter estimation and how the covariance matrix for each model can be calculated, see Bensmail and Celeux (1996).

Once the final converged estimates of the model have been obtained, these maximize the observed-data likelihood (Equation 7). The resulting parameters and $\hat{z}_{mg}$ values form the updated classification rule.

4. Where our package fits into existing software

There are some existing R packages that offer functionality for semi-supervised classfication. One such package is **bgmm**, written by law Biecek, Szczurek, Vingron, and Tiuryn (2012), which has parallels with the **upclass** functionality in that it can be used for mixture models and it also allows for restrictions on the covariance matrices. The **bgmm** models are codified with four-letter strings, the first of which relates to the mean vectors of the components, which can be the same or different. **mclust** does not do this, so the first letter of the string does not appear in our comparison in table 2. The second and third letters relate to the *between* covariance and *within* covariance matrices of the groups, which can either be equal (`"E"`) or different (`"D"`). The fourth letter letter relates to the covariances in each covariance matrix which can all be forced to be zero (`"0"`) or different (`"D"`). We can summarise the models offered in the two packages in table 2.

upclass	**bgmm**
EII	E E 0
VII	D E 0
EEI	E D 0
VEI	
EVI	
VVI	D D 0
EEE	E D D
EEV	
VEV	
VVV	D D D
	E E D
	D E D

Table 2: Comparison of models available in **upclass** and **bgmm**.

Six of the **bgmm** models are handled by **upclass** as they are equivalent to existing models in **mclust**. Moreover, **mclust** provides four additional covariance matrix structures.

bgmm offers two interesting diagonal models, not proposed by **mclust**, which are highly parsimonious, and take advantage of compound symmetry covariance.

Other packages which provide support for semi-supervised classification are **spa**, written by Culp (2011) and **phyclust**, written by Chen (2011). However, **spa** is designed for graph-based estimation and linear regression, and not for mixture models. **phyclust** does propose a mixture model, but the package is specifically geared towards DNA sequence data, rather than towards food data.

Therefore, we believe our package adds something new in the arena of semi-supervised classification. For completeness, we have included some supervised classification functions, for comparison purposes but they are not the main focus of the package.

5. The software

In this section we will discuss how to use the package **upclass** in R (R Development Core Team 2011). It can be implemented in semi-supervised mode or in supervised mode. It makes extensive use of the package **mclust** (Fraley *et al.* 2012). If **mclust** is not installed, **upclass** will install it.

The package is available on CRAN and can be installed using the following code.

```
 [1]  1.26461167 -0.61670573 -0.01859692 -1.39628658  0.15991189 -0.13398282
 [7]  0.27331773  1.91194226 -0.68619384  0.56588094
```

```
R> install.packages("upclass")
R> library("upclass")
```

5.1. Summary of functions available in the package

The R package **upclass** contains the following functions. The use of these will be outlined in the following sections. The most important of these is `upclassify()` and this will be described in most detail.

- The function `Aitken()` calculates the Aitken acceleration estimate of the final converged maximized log-likelihood.
- The function `modelvec()` list the valid model names to be used in the **upclass** package for univariate and multivariate data
- The function `noupclassify()` is used to carry out supervised classification over a range of different models and find the model that best fits the data.
- The function `noupclassifymodel()` is an internal work function used by `noupclassify`.
- The function `plot.upclassfit()` is a method used to produce a plot of the best model found by `upclassify` or `noupclassify`.
- The function `print.upclassfit()` is the `print` method for the user defined class used by **upclass**.

- The function `summary.upclassfit()` is the `summary` method for the user defined class used by **upclass**.
- The function `upclassify()` is the main function in the package. It is used to carry out semi-supervised classification over a range of different models and find the model that best fits the data.
- The function `upclassifymodel()` is an internal work function used by `upclassify`.

5.2. Setting up the data

To illustrate the use of the functions in **upclass**, we will employ the olive oil dataset (Forina *et al.* 1983). The dataset is found in the **classifly** package developed by Wickham (2011). We will set up the data in the following way.

```
R> data("olives")
R> set.seed(11)
R> X <- as.matrix(olives[,-c(1:2)])
R> cl <- olives[,1]
R> N <- dim(X)[1]
R> indtrain <- sort(sample(1:N, N * 0.2))
R> Xtrain <- X[indtrain,]
R> cltrain <- cl[indtrain]
R> indtest <- setdiff(1:N, indtrain)
R> Xtest <- X[indtest,]
R> cltest <- cl[indtest]
```

This assigns 20% of the data (114 observations) as labeled data (`Xtrain`, `cltrain`), where `Xtrain` are the observations and `cltrain` are their labels. The remaining data (458 observations) are to be thought of as unlabeled. The observations are stored as `Xtest`, and their removed labels as `cltest`.

Note that we deliberately set the seed at 11, as this is a tricky cut of the data. Setting the seed at 1, for example, gives excellent classification results from both supervised and semi-supervised methods.

5.3. The function `Aitken()`

This internal function takes in a vector of three consecutive log-likelihoods and estimates the final converged maximized log-likelihood using the method of Aitken acceleration described by Böhning, Dietz, Schaub, Shlattmann, and Lindsay (1994).
The calling functions can then decide if the log-likelihood has converged based on some specified tolerance, defaulted to 10^{-5} in the case of the **upclass** functions. When using `Aitken()`, specify the following argument.

- `ll` A vector of three consecutive log-likelihoods.

In practice, Aitken acceleration needs three values of the log-likelihood, so at the start of the algorithm, the calling function should initialise the three components of `ll` to $-\infty$, and each iteration of the algorithm should update the vector by moving the components along. This results in a minimum of three iterations being required for any algorithm using Aitken acceleration as a convergence tool.

This function could, of course, be used by itself, as in the following snippet.

```
R> ll <- c(-261, -257.46,-256.4)
R> Aitken(ll)

$ll
[1] -256.4

$linf
[1] -254.8869

$a
[1] 0.299435
```

`ll` gives the current estimate for the log-likelihood, while `linf` gives the estimate of the converged value and if the difference between the two should be less than some specified tolerance, convergence can be said to have been reached.

5.4. The function `modelvec()`

The internal function `modelvec()` stores the valid **mclust** models in two character vectors, the first for univariate data and the second for multivariate data for use by other **upclass** functions. When using `modelvec()`, specify the following argument.

- `d` The dimension of the data being used.

If `d` is 1, the data is considered to be univariate, and a vector containing the two univariate models handled by **mclust** are returned. Otherwise, the vector will contain the ten multivariate **mclust** models, as shown.

```
R> modelvec(1)

[1] "E" "V"

R> modelvec(2)

 [1] "EII" "VII" "EEI" "VEI" "EVI" "VVI" "EEE" "EEV" "VEV" "VVV"
```

5.5. The function `upclassify()`

The function `upclassify()` is used to classify unlabeled data by the semi-supervised method described in section 3.1. It produces estimates for the labels of the data, as well as model parameters for the models requested. In addition, it provides quantifiable goodness of fit measures as will be described below. Below is a list of `upclassify`'s arguments. Note that the first three must always be included. The remainder are optional.

- `Xtrain` The labeled data - A numeric matrix of data where rows correspond to observations and columns correspond to variables. The group membership of each observation is known.
- `cltrain` A numeric vector with distinct entries representing a classification of the corresponding observations in `Xtrain`
- `Xtest` The unlabeled data - A numeric matrix of data where rows correspond to observations and columns correspond to variables. The group membership of each observation will usually not be known.
- `cltest` An optional numeric vector with distinct entries representing a classification of the corresponding observations in `Xtest`. By default, these are not supplied and the function sets out to obtain them.
- `modelscope` A character string indicating the desired models to be tested. With default `NULL`, all available models are tested. The models available for univariate and multivariate data can be retrieved using function `modelvec()`.
- `tol` The tolerance required for convergence. The default is 10^-5.
- `iterlim` The maximum number of iterations if convergence has not been reached. The default is 1000.
- `Aitken` Whether or not Aitken acceleration is to be used. The default is set to `TRUE`. A simpler convergence criterion can be used by setting `Aitken` to `FALSE`. This method achieves convergence when two successive values of `ll` have a difference smaller than `tol`. The simpler convergence criterion has been shown to be less strict than the Aitken one (McNicholas, Murphy, McDaid, and Frost 2010).

The function output for each model comprises a list of sublists, one for each model tried, and the best one, using the BIC criterion (Schwarz 1978) , which is output first. Each sublist can be accessed by its model name, and the best one by the name `"Best"`.

In the interests of brevity, we will list each entry in turn instead of showing the actual output.

- `$call` How to call the function and the order of its arguments.
- `$Ntrain` The number of observations in the training set.
- `$Ntest` The number of observation in the test set.
- `$d` The dimension of the data.
- `$G` The number of groups in the training data. The package will not try to assign the test data to any other groups.
- `$iter` The number of iterations taken.
- `$converged` Whether or not the algorithm has converged. If `$converged` is `FALSE`, then `$iter` will be the maximum number of iterations (specified in the function call).
- `modelName` The model considered in this run of the algorithm.

- `$parameters` A list of the final model parameters estimated by the algorithm.
  - `$parameters$pro` The proportion of the data found to be in each group.
  - `$parameters$mean` Mean vectors for each group.
  - `$parameters$variance` A comprehensive list of variances and covariance matrices produced by **mclust**
- `$train` A list of information about the training data. This will not have changed from before the run.
  - `$train$z` A matrix containing the estimated probabilities that each observation in the training data belongs to each group.
  - `$train$cl` A vector containing the new labels of the training data.
  - `$train$misclass` The number of misclassifications of the training data.
  - `$train$rate` The misclassification rate expressed as a percentage.
  - `$train$Brierscore` The Brier score for the training data.
  - `$train$tab` The misclassification table by group.
- `$test` A list of information about the test data.
  - `$test$z` A matrix containing the estimated probabilities that each observation in the test data belongs to each group.
  - `$test$cl` A vector containing the estimated labels of the test data. This derives from the z matrix. Each observation is assigned to the group with the highest probability.
  - `$test$misclass` The number of misclassifications of the test data.
  - `$test$rate` The misclassification rate expressed as a percentage.
  - `$test$Brierscore` The Brier score for the test data. See below for an explanation of the Brier score.
  - `$test$tab` The misclassification table by group.
- `$ll` The log likelihood of the data.
- `$bic` The Bayes Information Criterion for the specified model.

In our example, we have the test labels to hand so we can include them in our model. The function will therefore produce classification performance results. If we omitted the `cltest` argument, only the model name would be returned, similar to **mclust** clustering output.

```
R> fitup <- upclassify(Xtrain, cltrain, Xtest,cltest)
R> fitup

Model Name:  VVV
Total Misclassified:  8
Misclassification Rate:   1.747 %
```

Note that the BIC criterion usually selects the model with the lowest misclassification rate, but not always. With this data split, the VVI model had no misclassifications but had a less satisfactory BIC than the VVV model. We can confirm this by interrogating the specific element of the output list.

```
R> fitup[["VVI"]]$test$misclass
```

```
[1] 0
```

We confirm that the VVI model had no misclassifications. By comparing `fitup[["Best"]bic` and `fitup[["VVI"]bic`, we can see that the BIC for the `[["Best"]]` model (VVV) is higher.

```
R> c(fitup[["Best"]]$bic,fitup[["VVI"]]$bic)
```

```
[1] -42866.66 -46763.30
```

This is why the VVV model was chosen, as BIC can always be calculated, whether or not we have the labels.

If we were interested in a full model list, for the VEV model for example, we could display it as follows.

```
R> fitup[["VEV"]]
```

We omit the output.

In order to classify our data for a smaller selection of models, we make use of the function `modelvec()`. If, for some reason, we were only interested in models with groups of equal size and shape, we could retrieve the **mclust** models EEI, EEE and EEV from `modelvec(2)`, and proceed to classify as follows.

```
R> models <- modelvec(2)
R> modelscope <- c(models[3],models[7],models[8])
R> fitupEE <- upclassify(Xtrain, cltrain, Xtest, cltest, modelscope=modelscope)
R> fitupEE
```

```
Model Name:  EEV
Total Misclassified:  19
Misclassification Rate:   4.148 %
```

How do we know it works?

Classification performance can be assessed using percentage misclassification error, provided we have the correct labels.

In our original example, which we recap here, we used the code

```
R> fitup <- upclassify(Xtrain, cltrain, Xtest,cltest)
R> fitup
```

```
Model Name:  VVV
Total Misclassified:  8
Misclassification Rate:   1.747 %
```

We can see that we got 8 observations misclassified, in this case, and a misclassification rate of 1.747%, but of course, we had the labels to hand. This would not generally be the case.

The Brier score is another useful indication of the accuracy of a probabilistic model prediction, proposed by Brier (1950). Brier's score gives an indication of how accurate the classifications are in terms of probability of group membership rather than just on the hard classification results. This is most useful when the correct labels are not available.

$$\text{Briers score } = \frac{100}{2M}\sum_{g=1}^{G}\sum_{m=1}^{M}(l^{mg} - z^{mg})^2,$$

where	l^{mg}	are the labels finally assigned to each observation.
and	z^{mg}	are their *a posteriori* probabilities.

A perfect prediction gives a Brier score of zero. If z^{mg} is a hard classification rather than the probability of observation m belonging to group g, then the Brier score becomes equivalent to the percentage misclassification error.

When classifying any dataset, some observations may be classified into the correct group, but the probability of membership of that group may not be much larger than the probability of membership of the other groups. Such observations con- tribute more to the Brier score than definitively correctly classified observations. Likewise, some observations may be incorrectly classified, but the probability of belonging to the correct group is not much lower than that of the chosen group. These observations contribute less to the Brier score than definitively incorrectly classified observations.

Overall, therefore, the Brier score can be used as an approximation to the misclassification rate, if the latter is not available.

The Brier score can be retrieved from the output as follows. For the example above:

```
R> fitup[["Best"]]$test$Brierscore
```

```
[1] 1.744615
```

This is a little lower than the misclassification rate of 1.747% indicating that the incorrectly classified observations have slightly higher uncertainty associated with them, whereas correctly classified observations have high probability of belonging to the correct group, (Toher *et al.* 2007).

For our other example, `fitupEE`, where we have restricted the classification to those models of equal size and shape, we get

```
R> fitupEE[["Best"]]$test$Brierscore
```

```
[1] 4.148472
```

We can see that this is very close to the misclassification rate.

5.6. The function `upclassifymodel()`

This is an internal function used by `upclassify()` and should not be used on its own. Full details of arguments and output are in the relevant R helpfile, if required.

5.7. The function `noupclassify()`

This function is included in the package for convenience. It allows the user to classify data based on a rule derived from the training set, while taking advantage of **mclust**'s covariance structures.

The arguments and the output list are the same as for `upclassify()` so will not be listed again. The `print()`, `summary()` and `plot()` methods are also common to both functions.

Still using the 20% labeling example, we can run `noupclassify()` as we would expect.

```
R> noupclassify(Xtrain, cltrain, Xtest,cltest)

Model Name:  VVV
Total Misclassified:  31
Misclassification Rate:   6.769 %
```

As before, if we do not have the labels, just the model chosen is returned by the `print()` method. The full output can be interrogated in the same way as for `upclassify`.

5.8. The function `noupclassifymodel()`

Like `upclassifymodel()`, this is an internal function and is not designed to be used on its own and full details of arguments and output are in the R helpfile, if required.

5.9. User defined methods for upclass

Both `upclassify` and `noupclassify` have output with the user-defined class of `upclassfit`. A `print()` and a `summary()` for this class of output has been developed. The `print()` has already been detailed for cases where we have labels for the test data. If we do not have these, the `print()` method only returns the selected model. Using our example above with the EE* models, we would get

```
R> fitupEEnl <- upclassify(Xtrain,cltrain,Xtest,modelscope=modelscope )
R> fitupEEnl

Model Name:  EEV
```

The output from `summary()` is slightly more extensive.

```
R> summary(fitupEEnl)
```

```
Model Name
 EEV
Log Likelihood
 -21517.02
Dimension
 8
Ntrain
 114
Ntest
 458
bic
 -43783.23
```

and with a little more if we have the labels.

```
R> summary(fitup)
```

```
Model Name
 VVV
Log Likelihood
 -21007.94
Dimension
 8
Ntrain
 114
Ntest
 458
bic
 -42866.66
Total Misclassified:  8
Misclassification Rate:   1.747 %
```

Any of the other output generated by `upclassify()` or `noupclassify()` can always be interrogated from the list. See Sections 5.5 and 5.7.

There is also a `plot()` function which shows the z values of the test data in graphical form. Problem points can easily be identified. See figure 2.

It should be noted that we left the label percentage at 20% but we changed the data set-up described in Section 5.2 to use `set.seed(4)`. This was to get plots with some less clearcut classifications. We used `fitup4` and `fitnoup4` as the names for our new fitted models to avoid ambiguity.

The plot for the `upclassify()` output follows.

```
R> fitup4 <- upclassify(Xtrain, cltrain, Xtest,cltest)
R> plot(fitup4)
```

We can see that most points are classified with a z_{ig} value of very close to 1 or 0, indicating a very high probability of belonging to a particular group, and a correspondingly low probability of belonging to any of the other groups, resulting in a straightforward classification decision.

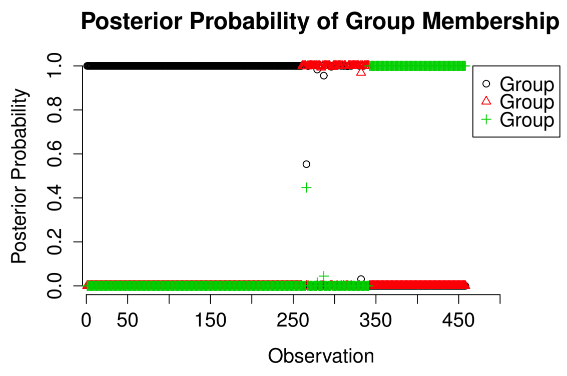


Figure 2: Z-values of a test subset of the olive oil data, using semi-supervised classification.

However if we look at observation 266 in the output's test z-matrix, we can see that its values are less than ideal, with similar posterior probabilities of being in either group 1 or group 3.

```
R> fitup4$Best$test$z[266,]

[1] 5.532143e-01 9.056811e-15 4.467857e-01
```

Although, `upclassify` will still classify this observation into group 1, the plot helps to show that this point may warrant further investigation.

For interest, we also supply a plot of the `noupclassify()` output for the same cut of the data. It shows a difficulty in classifying the group 2 data, using this method.

```
R> fitnoup4 <- noupclassify(Xtrain, cltrain, Xtest,cltest)
R> plot(fitnoup)
```

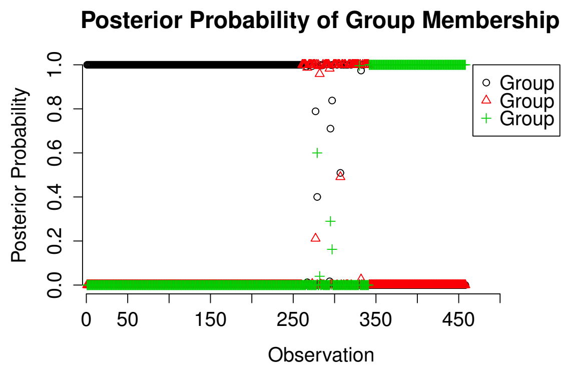


Figure 3: Z-values of the same subset of the olive oil data, using supervised classification.

5.10. Comparing our method with supervised classification (using mclust)

We compare how our method performs compared to classification rules formed using the training data only using the olive oil data set (Forina *et al.* 1983) as an example. This data set is made up of eight variables which measure the percentage composition of eight fatty acids on 572 olive oils. Each oil is from one of three Italian regions; North Italy, South Italy and Sardinia. Note that we did not classify by geographical area within region, neither did we include this finer geographical area variable in our training/test data.

To carry out this comparison, we created a function called `noupclassify()` which uses the `mstep()` command in **mclust** and loops through the selected covariance structures.

The data were randomly split into training (labeled) and test (unlabeled) data with a given percentage being assigned as training data. This process was repeated 200 times and each time, we classified the split of the data using both methods. In each case, the best model was selected using BIC and the classification performance was recorded for this model.

The code for this simulation test appears in the appendix.

Table 3 shows the results of classification tests using model-based discriminant analysis and the updated method, and also the number of times each model was selected at each level. It can be seen that VVV is selected as the best model until only 15% of the data are labeled. The VVV model, as the one with the most parameters, requires the training data to be of a certain size to work efficiently. On some data splits where fewer than 15% of the observations are labeled, some groups had insufficient observations to fit the VVV model so simpler models were selected; in these cases the selected models enumerated by the numbers in brackets. The variance of the misclassification rates is displayed in square brackets.

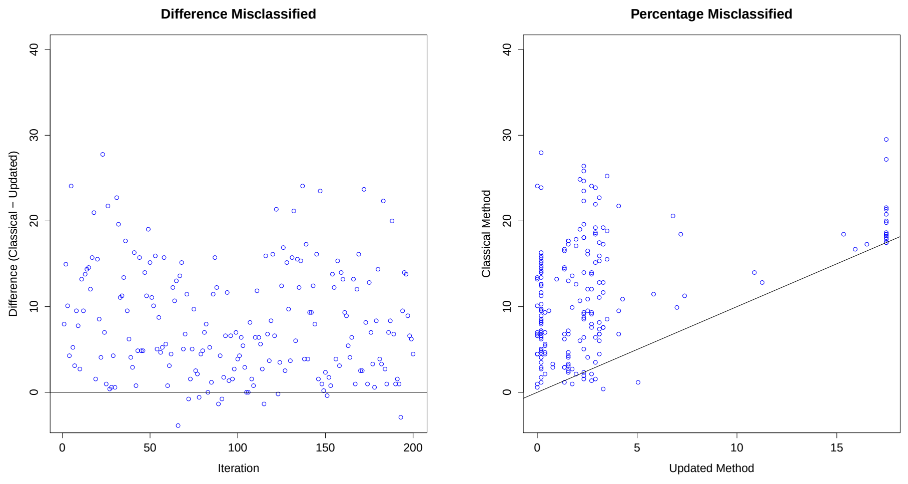
Labeled data	Classical method		Models		Updated method		Models	
90%	0.08	[0.128]	VVV		0.04	[0.073]	VVV	
80%	0.10	[0.077]	VVV		0.07	[0.056]	VVV	
70%	0.15	[0.093]	VVV		0.08	[0.041]	VVV	
60%	0.19	[0.093]	VVV		0.09	[0.036]	VVV	
50%	0.32	[0.174]	VVV		0.12	[0.064]	VVV	
40%	0.45	[0.230]	VVV		0.16	[0.095]	VVV	
30%	0.79	[0.531]	VVV		0.25	[0.239]	VVV	
20%	2.24	[5.474]	VVV		0.57	[0.989]	VVV	
15%	5.04	[19.06]	VVV	(195)	1.29	[6.541]	VVV	(194)
			VEV	(2)			VEV	(6)
			EEE	(3)				
10%	11.46	[47.01]	VVV	(141)	3.30	[23.44]	VVV	(137)
			VEV	(42)			VEV	(59)
			EEE	(17)			EEV	(10)

Table 3: Olive oil data: The percentage of labeled data versus the average misclassification rate (reported as a percentage) for the classical and updated methods. The frequency that each covariance structure was selected is also shown.

The results in Table 3 show the updated method outperforming the classical method at *every* level, although the methods show comparable results when more than 30% of the data are labeled. However, at the 30% level and lower, the results from the updated method far surpass those from the classical method.

At the 10% level, the difference between the two methods is very apparent. In this case, 515 observations are unlabeled out of the total of 572. The classical method misclassified nearly 11.5% of the unlabeled observations, which corresponds to 59 observations. The updated method has misclassified only 3.3% of the unlabeled data, which corresponds to only 17 observations.

Figure 4 compares the results of the two methods for each of the 200 iterations at the 10% level of labeled data.



(a) (b)

Figure 4: Olive oil data: Comparing the classical and updated methods at the 10% level: (a) shows the difference in the number of misclassified observations for the classical and updated methods. (b) shows a scatter plot of the number of misclassified observations for the classical and updated methods.

Figure 4(a) shows that the updated method almost always gives better results than the classical method. The updated method yielded a lower misclassification rate on 188 iterations out of 200, the same misclassification rate on 3 iterations and an inferior misclassification rate on 9 iteration. Figure 4(b) shows the number of observations that were misclassified by each method on each iteration. On 24 iterations, the classical method misclassified at least 20% of the unlabeled observations reaching a maximum of misclassifying almost 30% of the unlabeled observations for one iteration.

6. Discussion

We have presented a problem in data classification that occurs frequently in many areas, namely to classify unlabeled observations when only in possession of a small amount of labeled data. As such it would be of interest to researchers in food science, medical diagnostics, botany, or any area where such a scenario is commonplace.

We have introduced an R package called **upclass** which goes some way to addressing this problem. As we saw in Section 3 we can take advantage of the complete-data (both labeled and unlabeled) to create a classifier. The package is an implementation of the method developed in Dean *et al.* (2006), and takes advantage of the EM algorithm functionality developed by Fraley *et al.* (2012) in the package **mclust**.

We have described the functions available in the package in Section 5. The user can use the function `upclassify` to classify his data over the full range of models described in Section 2.1, or to select one or more suitable models. It is possible to vary the parameters to control convergence criteria, and also control the output produced. The function `noupclassify` is provided to carry out supervised classification, if desired. Functions are provided to interrogate the output from the functions in the package.

We have shown (in Table 3) that this method can provide better results at low levels of labeling than supervised classification.

The main limitation of the idea is that we assume that each group can be modeled by a normal distribution as we discussed in Section 2.1; in some applications this assumption may not be appropriate. Also, at very low levels of labeling, **upclass** cannot fit all possible models for the data, and must choose among the models with a suitably reduced number of parameters where model fitting is feasible.

The package is currently based on the ten covariance structures in **mclust** however fourteen covariance structures are possible within the modified eigen-decomposition. It would be interesting to extend the approach to all fourteen covariance structures (Biernacki *et al.* 2006) to increase the flexibility of this approach further.

A possible avenue for exploration, is to cater for the situation where not all groups are represented in the training set. This becomes more and more likely at very low levels of labeling, and might have other applications if any new datapoint belongs to a previously unknown group.

Acknowledgements

The authors would like to thank the associate editor and two anonymous referees for their helpful remarks and suggestions that contributed to a better presentation of the paper.

This research is supported by the Programme for Research In Third Level Institutions (PRTLI) Cycle 5 and co-funded by the European Regional Development Fund, and the Science Foundation Ireland Research Frontiers Programme (2007/RFP/MATF281).

A. Simulation Code

```
R> library(mclust)
R> library(classifly)
R> set.seed(1)
R> prop <- 0.9  #change this to vary proportions of labelled data.
```

```
R> data(olives)
R> runlength <- 200

R> nomodel <- vector(mode="character", length=runlength)
R> model <- vector(mode="character", length=runlength)
R> ratevec <- vector(mode="numeric", length=runlength)
R> noratevec <- vector(mode="numeric", length=runlength)

R>  for(j in c(1:runlength))
R>  {
R>  X <- as.matrix(olives[,-c(1,2)])
R>  cl <- unclass(olives[,1])
R>  N <- dim(X)[1]
R>  indtrain <- sort(sample(1:N,N * prop))
R>  Xtrain <- X[indtrain,]
R>  cltrain <- cl[indtrain]
R>  indtest <- setdiff(1:N, indtrain)
R>  Xtest <- X[indtest,]
R>  cltest <- cl[indtest]

R>  fitnoup <- noupclassify(Xtrain,cltrain,Xtest,cltest)
R>  noratevec[j] <- fitnoup[["Best"]]$test$rate
R>  nomodel[j] <- fitnoup[["Best"]]$modelName

R>  fitup <- upclassify(Xtrain,cltrain,Xtest,cltest)
R>  ratevec[j] <- fitup[["Best"]]$test$rate
R>  model[j] <- fitup[["Best"]]$modelName
R> }
R> list1 <- list()
R> list1$percentage <- pcentvec[i]*100
R> list1$notable <- table(nomodel)
R> list1$noratevec <- noratevec
R> list1$table <- table(model)
R> list1$ratevec <- ratevec

R> list1$var1 <- var(ratevec)
R> list1$novar1 <- var(noratevec)
R> list1
```

References

Banfield JD, Raftery AE (1993). "Model-Based Gaussian and Non-Gaussian Clustering." *Biometrics. Journal of the Biometric Society*, **49**(3), 803–821.

Bensmail H, Celeux G (1996). "Regularized Gaussian Discriminant Analysis through Eigenvalue Decomposition." *Journal of the American Statistical Association*, **91**(436), 1743–1748.

Biernacki C, Celeux G, Govaert G, Langrognet F (2006). "Model-Based Cluster and Discriminant Analysis with the **MIXMOD** Software." *Computational Statistics & Data Analysis*, **51**(2), 587–600.

Biernacki C, Govaert G (1999). "Choosing Models in Model-Based Clustering and Discriminant Analysis." *Journal of Statistical Computation and Simulation*, **64**, 49–71.

Böhning D, Dietz E, Schaub R, Shlattmann 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**(2), 373–388.

Brier GW (1950). "Verification of Forecasts Expressed in Terms of Probability." *Monthly Weather Review*, **78**(1), 1–3.

Caetano S, Üstün B, Hennessy S, Smeyers-Verbeke J, Melssen W, Downey G, Buydens L, Vander Heyden Y (2007). "Geographical Classification of Olive Oils by the Application of CART and SVM to their FT-IR." *Journal of Chemometrics*, **21**(7-9), 324–334.

Chapelle O, Schölkopf B, Zien A (eds.) (2006). *Semi-Supervised Learning*. MIT Press, Cambridge, MA.

Chen W (2011). *Overlapping Codon Model, Phylogenetic Clustering, and Alternative Partial Expectation Conditional Maximisation Algorithm*. Ph.D. thesis, Iowa State University. URL `http://thirteen-01.stat.iastate.edu/snoweye/phyclust`.

Culp M (2011). "**spa**: A Semi-Supervised R Package for Semi-Parametric Graph-Based Estimation." *Journal of Statistical Software*, **40**(10), 1–29. URL `http://www.jstatsoft.org/v40/i10/`.

Dean N, Murphy TB, Downey G (2006). "Using Unlabelled Data to Update Classification Rules with Applications in Food Authenticity Studies." *Journal of the Royal Statistical Society. Series C. Applied Statistics*, **55**(1), 1–14.

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), 1–38. With Discussion.

Fan Y, Murphy TB, Byrne J, Brennan L, Fitzpatrick J, Watson RWG (2011). "Applying Random Forests to Identify Biomarker Panels in Serum 2D-DIGE Data for the Detection and Staging of Prostate Cancer." *Journal of Proteome Research*, **10**(3), 1361–1373.

Forina M, Armanino C, Lanteri S, Tiscornia E (1983). "Classification of Olive Oils from Their Fatty Acid Composition." In H Martens, H Russwurm Jr (eds.), *Food Research and Data Analysis*, pp. 189–214. Applied Science Publishers, London.

Fraley C, Raftery A (2007). "Model-Based Methods of Classification: Using the **mclust** Software in Chemometrics." *Journal of Statistical Software*, **18**(6), 1–13.

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

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.

Ganesalingam S, McLachlan GJ (1978). "The Efficiency of a Linear Discriminant Function Based on Unclassified Initial Samples." *Biometrika*, **65**(3), 658–662.

Joachims T (1999). "Transductive Inference for Text Classification using Support Vector Machines." In *ICML '99: Proceedings of the Sixteenth International Conference on Machine Learning*, pp. 200–209. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. ISBN 1-55860-612-2.

Kass R, Raftery AE (1995). "Bayes Factors and Model Uncertainty." *Journal of the American Statistical Association*, **90**, 773–795.

law Biecek P, Szczurek E, Vingron M, Tiuryn J (2012). "The R Package **bgmm**: Mixture Modeling with Uncertain Knowledge." *Journal of Statistical Software*, **47**(3).

McLachlan G (1992). *Discriminant Analysis and Statistical Pattern Recognition*. Wiley, New York.

McLachlan GJ (1975). "Iterative Reclassification Procedure for Constructing an Asymptotically Optimal Rule of Allocation in Discriminant Analysis." *Journal of the American Statistical Association*, **70**, 365–369.

McLachlan GJ (1977). "Estimating the Linear Discriminant Function from Initial Samples Containing a Small Number of Unclassified Observations." *Journal of the American Statistical Association*, **72**(358), 403–406.

McNicholas PD (2010). "Model-Based Classification using Latent Gaussian Mixture Models." *Journal of Statistical Planning and Inference*, **140**(5), 1175–1181.

McNicholas PD, Murphy TB, McDaid AF, Frost D (2010). "Serial and Parallel Implementations of Model-Based Clustering via Parsimonious Gaussian Mixture Models." *Computational Statistics & Data Analysis*, **54**(3), 711–723.

Murphy TB, Dean N, Raftery AE (2010). "Variable Selection and Updating in Model-Based Discriminant Analysis for High Dimensional Data with Food Authenticity Applications." *Annals of Applied Statistics*, **4**(1), 396–421.

Nigam K, McCallum A, Mitchell T (2006). "Semi-Supervised Text Classification Using EM." In O Chapelle, B Schölkopf, A Zien (eds.), *Semi-Supervised Learning*. MIT Press, Boston.

O'Neill TJ (1978). "Normal Discrimination with Unclassified Observations." *Journal of the American Statistical Association*, **73**(364), 821–826.

Pouteau R, Meyer JY, Taputuarai R, Stoll B (2012). "Support Vector Machines to Map Rare and Endangered Native Plants in Pacific Islands Forests." *Ecological Informatics*, **9**, 37–46.

R Development Core Team (2011). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL `http://www.R-project.org/`.

Ripley BD (1996). *Pattern Recognition and Neural Networks*. Cambridge University Press.

Schwarz G (1978). "Estimating the Dimension of a Model." *The Annals of Statistics*, **6**(2), 461–464.

Toher D, Downey G, Murphy TB (2007). "A Comparison of Model-Based and Regression Classification Techniques Applied to Near Infrared Spectroscopic Data in Food Authentication Studies." *Chemometrics and Intelligent Laboratory Systems*, **89**(2), 102–115.

Toher D, Downey G, Murphy TB (2011). "Semi-Supervised Linear Discriminant Analysis." *Journal of Chemometrics*, **25**(12), 621–630.

Wang Y, Chen S, Zhou ZH (2012). "New Semi-Supervised Classification Method Based on Modified Cluster Assumption." *IEEE Transactions on Neural Networks and Learning Systems*, **23**(5), 689–702.

Wickham H (2011). *classifly: Explore Classification Models in High Dimensions*. R package version 0.3, URL `http://CRAN.R-project.org/package=classifly`.

Zhu X, Goldberg AB (2009). "Introduction to Semi-Supervised Learning." *Synthesis Lectures on Artificial Intelligence and Machine Learning*, **3**(1), 1–130.

Affiliation:

Niamh Russell
Complex and Adaptive Systems Laboratory
& School of Mathematical Sciences
Belfield Office Park
Clonskeagh
Dublin 4.
E-mail: `niamh.russell.1@ucdconnect.ie`